

VisualAge[®] C++ Professional for AIX[®]



IBM[®] Open Class[™] : User Interface

Version 5.0

Note!

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

Edition Notice

This edition applies to Version 5.0 of IBM VisualAge 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.

© Copyright International Business Machines Corporation 1998, 2000. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Programming Interface Information	ix
Trademarks and Service Marks	ix
Industry Standards	x

About This Book.	xi
-----------------------------------	-----------

Chapter 1. Windows **1**

Window Relationships	2
Create a Frame Window	4
Change the Title Bar	6
Create an Information Area	9
Create a Message Box	11

Chapter 2. Styles **15**

Add Styles	15
----------------------	----

Chapter 3. Events and Event Handlers **19**

Event and Event Handler Mechanism	23
Write an Event Handler	25
Extend Event Handling	27
Handle Mouse Events	33
Add Mouse Event Handling	36
Mouse Handler Example	40
File: mhsamp.cpp	40
File: mhsamp.h	42
File: mhsamp.hpp	43
File: mhsamp.rc	44
Set Time Intervals	44
Add Time Intervals	47
Monitor Text-Change Events	49
Use Keyboard Accelerators	50
Handle Text-Change Events	52

Chapter 4. Menus **55**

Add Menus to Your Application	57
Add a Menu Bar	59
Create a Pop-Up Menu	62
Create a Pop-Up Menu for a List Box	65
Pop-Up Menu Example	69
File: popmenu.cpp	69
File: popmenu.h	72
File: popmenu.hpp	73
File: popmenu.rc	74
Dynamically Modify a Menu with the ISubmenu Class	75

Chapter 5. Use Cursor Classes **79**

Chapter 6. Static Text Controls **81**

Create a Static Text Control	81
--	----

Chapter 7. Entry Fields and Multiline

Edit (MLE) Fields **83**

Create an Entry Field	83
Entry Field Example	87
File: entryf3.cpp	87
File: entryf3.h	89
File: entryf3.hpp	90
File: entryf3.rc	91
Create an MLE	92

Chapter 8. Buttons **97**

Create a Push Button	99
Create a Radio Button	103
Radio Button Example	107
File: radio.cpp	107
File: radio.h	110
File: radio.hpp	110
File: radio.rc	111
Create a Check Box	112
Check Box Example	114
File: checkbo1.cpp	114
File: checkbo1.h	115
File: checkbo1.hpp	116
File: checkbo1.rc	117
Create a Three-State Check Box	117

Chapter 9. List Boxes **119**

Create a List Box	119
List Box Example	123
File: listbox.cpp	123
File: listbox.h	126
File: listbox.hpp	126
File: listbox.rc	127
Add or Delete a List Box Item	128

Chapter 10. Combination Boxes **133**

Create a Combination Box	133
------------------------------------	-----

Chapter 11. Sliders **135**

Create a Slider Control	136
Slider Example	140
File: slider.cpp	140
File: slider.h	144
File: slider.hpp	144

Chapter 12. Spin Buttons **147**

Create a Spin Button	147
Spin Button Example	150
File: spinbtn.cpp	150
File: spinbtn.h	152
File: spinbtn.hpp	153

Chapter 13. Canvas Controls. **155**

Set Canvases	156
------------------------	-----

Create a Set Canvas	157
Set Canvas Example	160
File: esetcv.cpp	160
File: esetcv.h	162
File: esetcv.hpp	162
File: esetcv.rc	163
Split Canvases	164
Create a Split Canvas	165
Split Canvas Example 1	170
File: splitcan.cpp	170
File: splitcan.h	173
File: splitcan.hpp	173
File: splitcan.rc	175
Split Canvas Example 2	176
File: esplitcv.cpp	176
File: esplitcv.h	177
File: esplitcv.hpp	177
File: esplitcv.rc	178
Multicell Canvases	179
Create a Multicell Canvas	181
Multicell Canvas Example 1	184
File: mcc.cpp	184
File: mcc.h	186
File: mcc.hpp	187
Multicell Canvas Example 2	188
File: emcelcv.cpp	188
File: emcelcv.h	190
File: emcelcv.hpp	191
File: emcelcv.rc	192
View Ports	193
Create a View Port	193
View Port Example	196
File: vp1.cpp	196
File: vp1.h	198
File: vp1.hpp	198
File: vp1.rc	199
Drawing Canvases	200

Chapter 14. Notebooks 203

Use Notebooks	204
Create a Notebook	208
Specify Notebook Styles	209
Add Pages to the Notebook	211
Remove Notebook Pages	213
Change Notebook Colors	214
Tab Controls	214
Create a Tab Control	216
Create an OS/2 Warp 4.0 Style Notebook	219

Chapter 15. Containers 223

Create and Use Containers	225
Create Container Objects	225
Add and Remove Container Objects	227
Share Objects Among Containers	229
Filter Container Objects	230
Sort Objects in a Container	232
Access Container Objects Using an Object Cursor	233
Change Views in a Container	235
Define the Details View Using Container Columns	236
Create a Pop-Up Menu in a Container	239
Use the Windows Native Container Control	240

Chapter 16. Toolbars 243

Create a Toolbar	245
Add a Toolbar	247
Create Toolbar Buttons	252

Chapter 17. Common Dialogs 261

Specify File Dialog Information	261
Create a File Dialog	263
Specify Font Dialog Information	266
Create a Font Dialog	267

Chapter 18. Fonts 271

Construct Fonts	271
---------------------------	-----

Chapter 19. Help Information. 275

Create Help Information.	276
Add Fly-Over Help	282

Chapter 20. Clipboard. 285

Add Clipboard Support	286
Sample: Add Clipboard Support	289

Chapter 21. Resources 297

Chapter 22. Client/Server Applications and Dynamic Data Exchange. 303

Establishing DDE Conversations	304
The DDE Framework Design	306
Add DDE Client Support	307
Add DDE Server Support	309

Chapter 23. Direct Manipulation 313

Use Default Direct Manipulation	316
Use Rendering Mechanisms and Formats	320
Use Drag Item Types	323
Enable Direct Manipulation for an Entry Field or MLE	324
Enable Direct Manipulation for a Container	325
Set and Query the Drag Operation	330
Set the Target Emphasis	331
Add Images to Drag Items	331

Chapter 24. Multimedia Devices 335

Create Master Devices	338
Play Audio Compact Discs	340
Audio CD Example	348
File: samp5.cpp	348
File: samp5.h	352
File: samp5.hpp	353
Create Audio Devices	354
Playing/Recording Waveform Example	365
File: samp8.cpp	365
File: samp8.h	370
File: samp8.hpp	371
File: samp8.rc	372
Playing/Recording MIDI Example	373
File: samp6.cpp	373
File: samp6.h	375
File: samp6.hpp	376

File: samp6.rc	378
Create Video Devices	378
Add Animated Buttons and Circular Sliders	385

Chapter 25. Bidirectional Language Support 389

Set and Query Default Bidirectional Attributes	391
Assign Bidirectional Attributes to Individual Windows	393

Chapter 26. Port and Convert Applications 395

Convert Application Resources	395
Port Applications with Frame Windows	397
Use Native Window Controls and 3D Borders	398
Support Double-Byte Character Set and Multiple Languages	399

Chapter 27. Work with the User Interface Samples 401

Hello World Version 1: Create a Main Window	402
Hello World Version 2: Add Resource Files and Frame Extensions	405
Hello World Version 3: Add Command Handlers and Menu Bars	412
Hello World Version 4: Adding Dialogs and Push Buttons	420
Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help	433
Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics	443
Task and Samples Cross-Reference Table	445

Notices

Note to U.S. Government Users Restricted Rights -- use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd.
1150 Eglinton Avenue East
Toronto, Ontario M3C 1H7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

® (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. ® Copyright IBM Corp. 1998, 2000. All rights reserved.

Programming Interface Information

Programming interface information is intended to help you create application software using this program.

General-use programming interface allow the customer to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification, and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and Service Marks

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

AIX
AS/400
DB2
CICS
C Set ++
IBM
Network Station
Object Connection
OS/2
OS/390
OS/400
Open Class
Operating System/2
Operating System/400
PowerPC 403
PowerPC 601
PowerPC 603
PowerPC 604
Presentation Manager
RS/6000
S/390
SAA
Systems Application Architecture
TeamConnection
VisualAge
WebSphere
Workplace Shell

Lotus, Lotus Notes, and Domino are trademarks or registered trademarks of the Lotus Development Corporation in the United States, or other countries, or both.

Tivoli Management Environment, TME 10, and Tivoli Module Designer are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

Encina and DCE Encina Lightweight Client are trademarks of Transarc Corporation in the United States, or other countries, or both.

Microsoft, Win32, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

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

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, or other countries, or both.

C-bus is a registered trademark of Corollary, Inc.

PC Direct is a registered trademark of Ziff Communicatoins Company and is used by IBM Corporation under license

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

Industry Standards

VisualAge C++ Professional for AIX, Version 5.0 supports the following standards:

- The C language is consistent with the International Standard C (ANSI/ISO-IEC 9899-1990 [1992]). This standard has officially replaced American National standard for Information Systems-Programming Language C (X3.159-1989) and is technically equivalent to the ANSI C standard. VisualAge C++ supports the changes adopted into the C Standard by ISO/IEC 9899:1990/Amendment 1:1994.
- The IBM Systems Application Architecture (SAA) C Level 2 language definition.
- The C++ language is consistent with the International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:1998).
- The ISO/IEC 9945-1:1990/IEEE POSIX 1003.-1990 standard.
- The X/Open Common Applications Environment Specifications, System Interfaces and Headers, Issue 4.

About This Book

The information in this PDF document is also available in the online help.

To find this information, or any topics listed in this document as Related Concepts, Related Tasks, or Related References, simply type the topic title into the search bar in the top frame of your browser in the online help.

For some topics, the suggested references may already be contained in this document. In such cases, there is a cross-reference to the page on which the related topic appears.

Chapter 1. Windows

A *window* is a generic term for a rectangular area on the screen that behaves in a well defined manner. All windows have basic properties (such as size, position, boundaries, and visibility), and behavior (such as the ability to receive mouse or keyboard input, or to paint when visible).

The `IWindow` class is the base class for all windows. This class lets you query and set properties shared by all windows. However, you rarely need to create an `IWindow` object. Instead, create objects of classes derived from `IWindow` to represent specific types of windows, such as an `IFrameWindow` object to represent a frame window. Create an `IWindow` object (for example, from a window handle) only when the specific type of presentation system window is either unknown or unimportant.

► AIX

An `IWindow` object is designed to represent a Motif widget.

Frame Windows

A *frame window* provides the basic structure and support for the elements of an application. You can create and manipulate frame windows with the `IFrameWindow` class. The frame window provides support for basic components and features such as the following:

- Title bar
- Minimize and maximize buttons
- Sizing border
- System menu and menu bar

The frame window also coordinates these basic elements with the *client window*.

You can associate a frame window with a set of *resources*. (Resources are user-interface components, such as text strings, menu bars, icons, bitmaps, and keyboard accelerator tables, that you define in a separate file.)

The client window contains application-specific material; it should be the reason why your frame window exists in the first place. Thus, your application should display or collect data from the user via the client window. The client window fills the *client area*, the portion of the frame window not occupied by the basic frame components or the frame extensions.

You can add *frame extensions* to the frame window. A frame extension is a rectangular region that you can add to the outside edge of the client area of a frame window. You can add controls into frame extensions. You can place a frame extension left, right, above, or below the client area of a frame window. The information area implemented by the `IInfoArea` class is an example of a frame extension.

Presentation System

The *presentation system* is the windowing system for the operating system.

Desktop Window

The *desktop window* is the window that represents the entire screen. The `IWindow::desktopWindow()` function returns an `IWindow` object that represents the desktop window.

Controls

A *control* is a specialized window that provides a general-purpose interface, such as buttons, entry fields, and list boxes.

AIX

The AIX platform uses the term *widgets* for controls.

An *aggregate* or a *compound control* is a window that is actually comprised of more than one widget or control. The following are examples of aggregates:

- View port: consists of a horizontal scroll bar, vertical scroll bar, clipping window, and a parent window for these controls.
- Combination box: consists of an entry field, a list box, and a parent window for these controls that may draw the drop-down button.

Input Focus

The input focus determines which window receives all keyboard input. Only one window at a time may have input focus. A window indicates that it has input focus through a visual cue. For example, when an entry field has the input focus, it displays an input cursor. When a push button has the input focus, it becomes highlighted.

The input focus does not determine which window receives mouse input. Instead, the presentation system uses the position of the mouse pointer to determine the window to send mouse input. First it checks whether the topmost window underneath the pointer accepts mouse input. If the window does, it sends the mouse message to that window. If not, it checks the window underneath that window and continues to the desktop window until it finds a window that accepts mouse input, and sends the mouse message there. Some controls, such as static text on some platforms, ignore mouse input.

RELATED CONCEPTS

“Window Relationships”

“Chapter 3. Events and Event Handlers” on page 19

“Chapter 21. Resources” on page 297

RELATED TASKS

“Create a Frame Window” on page 4

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Window Relationships

Window relationships define how windows interact with each other. There are two kinds of relationships:

- Parent-child relationship
- Ownership

Parent-Child Relationship

The *parent-child relationship* determines the visual relationship between windows on the screen. The presentation system positions a *child window* relative to its *parent window*.

The presentation system draws child windows on top of their parent window, but the child windows cannot visually extend beyond the bounds of their parent window. The parent window crops its child windows to fit within its boundaries.

Typically frame windows have the desktop window as their parent.

OS/2

Frame windows that do not have the desktop as their parent are *child frame windows*. The Open Class Library supports child frame windows only on OS/2®. You specify the parent of a frame window when you construct the `IFrameWindow` object. Creating a frame window without a parent window is equivalent to creating it with the desktop window as its parent.

Windows that share the same parent window are *sibling windows*. The order the IBM Open Class creates sibling windows is called the *sibling order*. By default, the IBM Open Class creates new windows at the bottom of the sibling order. The sibling order defines the following:

- The order of painting (from bottom to top)
- The order of keyboard movement with the Tab key (top to bottom)
- The window to receive mouse events (the one on top)

Ownership

Ownership defines a path for message processing within the presentation system. A window always notifies its *owner window* of any significant events. For example, a button notifies its owner whenever a user selects it. A window also passes keyboard and mouse events that it does not process to its owner window for possible processing.

Ownership also defines the path for inheriting colors and fonts. For example, if a window doesn't have a specific font assigned to it, it uses the font of its owner window.

We recommend that you keep parent and owner windows the same.

OS/2

You must create a window and its owner window on the same thread.

There are two kinds of frame windows depending on ownership:

- *Primary windows*: Frame windows that have no owner or are owned by the desktop window.
- *Secondary windows*: Frame windows with an owner other than the desktop window.

Between two frame windows, ownership defines the following behavior:

- Closing a frame window returns the input focus to its owner window.

- Moving, minimizing, restoring, or closing a frame window causes the secondary windows it owns to move, minimize, restore, or close.

OS/2

You may specify the style `IFrameWindow::noMoveWithOwner` so that secondary windows do not move with their owner.

- Frame windows always remain above their owners.

You generally use a secondary window for the following purposes:

- To display views of subcomponents of objects displayed in your main view. For example, a list of properties and attributes of a file is a secondary window.
- To prompt the user for input as a result of selecting a menu choice. For example, the file dialog that appears after selecting the menu choice **Open** is a secondary window.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Frame Window”

“Add Styles” on page 15

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create a Frame Window

Use the `IFrameWindow` class to create a frame window. The default style of the `IFrameWindow` class has a title bar, system menu, minimize button, maximize button, and restze border. The default style adds an entry for the frame window to the system window list.

AIX

In Motif, the default style of the `IFrameWindow` class has a title bar, window menu button, minimize button, maximize button, and resize window border. The window menu button on AIX is a system menu button on OS/2. The Open Class Library uses the `IFrameWindow::systemMenu` style to create the window menu button for AIX.

The following sample code creates a default frame window:

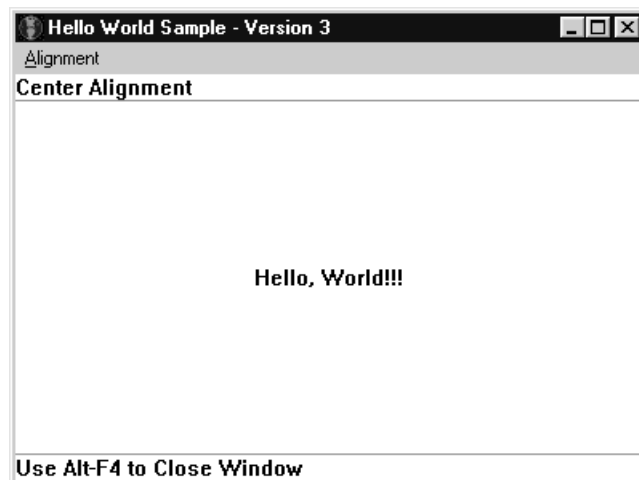
```
// A simple frame window example
#include <iframe.hpp>
#include <iapp.hpp>
#define MAIN_WINDOW 1
// Main Routine - application entry point
int main()
{
    // Construct the frame window
    IFrameWindow frame("Basic Frame Window",
                      MAIN_WINDOW,
                      IFrameWindow::defaultStyle());
    frame.setFocus().show();
    // Start event processing
    IApplication::current().run();
    return 0;
}
```


The following figure shows the frame window on the Windows® platform created using the preceding example:



The IFrameWindow class also provides several other styles. You can, for example, associate an accelerator key table to the frame window or provide an icon to be used when the window is minimized.

The following shows the components of a frame window created using the IFrameWindow class with the default style and some added controls. This image is from the Windows platform:



When you construct an IFrameWindow with a style of IFrameWindow::minimizedIcon, IFrameWindow::accelerator, or IFrameWindow::menuBar, resources corresponding to the style must be in the resource library you use to construct the frame. This library is usually the default user library, which you can use by using the following line of code:

```
Application::current().userResourceLibrary();
```

You can explicitly specify the resource library on the IFrameWindow constructor by using the const IResourceId argument, giving an IResourceLibrary value.

When you construct an `IFrameWindow` with a style of `minimizedIcon`, `accelerator`, or `menuBar`, the corresponding resource defined with the `IFrameWindow` window ID in the resource file is automatically loaded. If a required resource is not found, an exception is thrown, and the frame window is not constructed.

OS/2

When a frame window is minimized, the frame window hides and draws its minimized icon. Sometimes other windows associated with the frame window are drawn on top of its icon. This occurs when the windows are children of the frame window but not the client window or frame extensions.

OS/2

To suppress the drawing of child windows that are not frame extensions of the client window when they are supposed to be minimized, add a handler to the frame window that detects when the frame is minimized, and hides these windows. The handler should make these windows visible when the frame is restored.

The Open Class Library defers positioning and sizing components of a frame window until the frame window shows. As a result, if you query the size and position of the frame window's client window or frame extensions, an accurate value is not returned until the frame window is shown.

The `IWindow::show` or `IFrameWindow::showModally` member functions automatically update the frame window.

You can force the frame window to update itself by calling the `IFrameWindow::update()` member function.

RELATED CONCEPTS

"Chapter 1. Windows" on page 1

"Chapter 21. Resources" on page 297

"Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Add Styles" on page 15

"Convert Application Resources" on page 395

"Write an Event Handler" on page 25

"Hello World Version 1: Create a Main Window" on page 402

"Hello World Version 3: Add Command Handlers and Menu Bars" on page 412

"Task and Samples Cross-Reference Table" on page 445

Change the Title Bar

The *title bar* is the area at the top of each frame window that contains a window title.

WIN

OS/2

If you do not provide a window title, the Open Class Library sets the title to a string loaded from the application's resource library. The ID of the string in the string table is the frame window's ID.

AIX

In Motif, if you do not provide a window title, Open Class Library looks for a string in the application's resource file. If the Open Class Library cannot find a string, the title defaults to the system-generated title (typically, the name of the executable file).

You can specify the icon which displays when the application is minimized. Use the `minimizedIcon` style when you create the frame window.

The following code, from Hello World version 3, shows you how to specify a minimized icon and the window title when you create the frame window.

1. Define the title text and the minimized icon in the resource script file:

```
#include "ahello3.h"
ICON WND_MAIN ahello3.ico
STRINGTABLE
BEGIN
    STR_HELLO, "Hello, World!!!"
    WND_MAIN, "Hello World Sample - Version 3"
// ...
```

AIX

OS/2

The above resource script file is for AIX or OS/2.

`WND_MAIN` is the frame window identifier. The frame window uses the window identifier (`windowId`) passed on the constructor to load its icon, title, menu bar, or accelerator table resources if these components are specified in the frame window style.

2. Use the `defaultStyle` and `minimizedIcon` styles in the `IFrameWindow` constructor. The following code comes from the `ahello3.cpp` file:

```
int main()
{
    // ...
    AHelloWindow mainWindow (WND_MAIN);
    mainWindow.setTextAlignment(AHelloWindow::left);
    mainWindow.setSize(400,300);
    mainWindow.setFocus();
    mainWindow.show();
    IApplication::current().run();
    return 0;
}
// ...
AHelloWindow :: AHelloWindow(const unsigned long windowId)
:IFrameWindow(IFrameWindow::defaultStyle() |
              IFrameWindow::minimizedIcon,
              windowId)
,menuBar(windowId, this)
,statusLine(WND_STATUS,this,this)
,hello(WND_HELLO, this, this)
,infoArea(this)
```

```

    ,commandHandler(this)
    {
        // ...
    }

```

When the application creates the AHelloWindow object, it constructs the IFrameWindow base class using the default style with a minimized icon, ahellow3.ico, and “Hello World Sample - Version 3” as the title text. The frame window determines the icon and title text based on the window ID and the resource library.

To access or change your frame window’s title, use the IFrameWindow::titleText and IFrameWindow::setTitleText member functions. You can also create an ITitle object.

The ITitle Class

The ITitle class creates and updates the title bar area of your frame window. The ITitle class consists of the following three components:

- Object text
- View text
- View number

When you construct a title, you must provide the object text. The other two components are optional. The Open Class Library separates the object text and view text with a hyphen (-). The library separates the view text and view number with a colon (:), as in the following example:

```
OS/2 System - Icon View:2
```

Use ITitle if you want the Presentation Manager-compatible support for the object, view, and view number. If you do not want this support, you can specify a title using an IFrameWindow constructor or calling the IFrameWindow::setTitleText function again.

WIN

ITitle does not create the title bar as a separate window on the Windows operating system. The setAutoDeleteObject function (which deletes the object when the window is deleted) does not work because no window is being deleted. Therefore, do not use the setAutoDeleteObject function to delete a title bar. Instead, perform your own delete for the objects that your application creates using the new operator.

OS/2

The title may exceed 60 bytes in length, but only the first 60 bytes appear in a Window List entry. Use IFrameWindow::addToWindowList to add entries to the window list.

AIX

An ITitle window is not a separate window owned by the application; it is created and managed by the Window Manager. As a result, you cannot use IWindow::setAutoDeleteObject to automatically delete ITitle objects. Instead perform your own delete for the ITitle objects that your application creates using operator new. Similarly, you cannot use IWindow::setAutoDestroyWindow.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Hello World Version 3: Add Command Handlers and Menu Bars” on page 412

“Create a Frame Window” on page 4

“Create an Information Area”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create an Information Area

The *information area* is a small rectangular area that is usually located at the bottom of a frame window. You can use the information area to display:

- A brief explanation of the state of an object
- Information about the completion of a process
- Information messages displayed with fly-over help

Use the `IInfoArea` class to create and manage the information area. Objects of `IInfoArea` class provide a frame extension to show information about the menu item where the cursor is positioned. The string displayed in the information area is defined in a string table in the resource file.

The following sample uses the `IInfoArea` class to create the information area and the text to display in it.

1. Define the menu bar and string table in a resource script file. The string table contains strings of text, and each string is associated with a menu item. For Windows, when you choose the menu item, the string related to that item displays in the information area. The following is from the `ahellow3.rc` file.

➤ AIX

➤ OS/2

The following is a sample OS/2 and AIX resource script file:

```
#include "ahellow3.h"
ICON WND_MAIN ahellow3.ico
STRINGTABLE
BEGIN
    STR_HELLO,    "Hello, World!!!"
    WND_MAIN,    "Hello World Sample - Version 3"
    STR_INFO,    "Use Alt-F4 to Close Window"
    MI_ALIGNMENT, "Alignment Menu"
    MI_CENTER,   "Set Center Alignment"
    MI_LEFT,     "Set Left Alignment"
    MI_RIGHT,    "Set Right Alignment"
    STR_CENTER,  "Center Alignment"
    STR_LEFT,    "Left Alignment"
    STR_RIGHT,   "Right Alignment"
END
MENU WND_MAIN
BEGIN
    SUBMENU "~Alignment", MI_ALIGNMENT
    BEGIN
        MENUITEM "~Left", MI_LEFT
```

```

        MENUITEM "~Center", MI_CENTER
        MENUITEM "~Right", MI_RIGHT
    END
END

```

WIN

The same resource file would appear as follows for a Windows application:

```

#include "ahellow3.h"
WND_MAIN ICON ahellow3.ico
STRINGTABLE
    BEGIN
        STR_HELLO,    "Hello, World!!!"
        WND_MAIN,    "Hello World Sample - Version 3"
        STR_INFO,    "Use Alt-F4 to Close Window"
        MI_ALIGNMENT, "Alignment Menu"
        MI_CENTER,   "Set Center Alignment"
        MI_LEFT,     "Set Left Alignment"
        MI_RIGHT,    "Set Right Alignment"
        STR_CENTER,  "Center Alignment"
        STR_LEFT,    "Left Alignment"
        STR_RIGHT,   "Right Alignment"
    END
WND_MAIN MENUEX
    BEGIN
        POPUP "&Alignment", MI_ALIGNMENT
        BEGIN
            MENUITEM "&Left", MI_LEFT
            MENUITEM "&Center", MI_CENTER
            MENUITEM "&Right", MI_RIGHT
        END
    END
END

```

- This code is from the ahellow3.hpp file. The highlighted lines add an information area object to the AHelloWindow class.

```

class AHelloWindow : public IFrameWindow
{
public:
    enum Alignment
    {
        left, center, right
    };
    AHelloWindow(const unsigned long windowId);
    virtual
        AHelloWindow();
    virtual AHelloWindow
        &setTextAlignment( const Alignment alignment);
private:
    IMenuBar
        menuBar;
    IStaticText
        statusLine,
        hello;
    IInfoArea
        infoArea;
    ACommandHandler
        commandHandler;
    AHelloWindow
        &operator= (const AHelloWindow&);
};

```

- In the ahellow3.cpp file, construct the information area when AHelloWindow is created as follows:

```

AHelloWindow :: AHelloWindow(const unsigned long windowId)
    : IFrameWindow(IFrameWindow::defaultStyle() |
        IFrameWindow::minimizedIcon,

```

```

        windowId)
,menuBar(windowId, this)
,statusLine(WND_STATUS,this,this)
,hello(WND_HELLO, this, this)
,infoArea(this)
,commandHandler(this)
{
    setClient(&hello);
    addExtension(&statusLine,
        IFrameWindow::aboveClient,
        IFont(&statusLine).maxCharHeight());
    hello.setText(STR_HELLO);
    infoArea.setDefaultText(STR_INFO);
    commandHandler.handleEventsFor(this);
    setTextAlignment(center);
}

```

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Hello World Version 3: Add Command Handlers and Menu Bars” on page 412

“Create a Frame Window” on page 4

“Create Help Information” on page 276

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create a Message Box

A *message box* is a frame window that an application uses to display a note, caution, or warning to the user. For instance, an application can use a message box to inform a user of a problem that the application encountered while performing a task. The Open Class Library provides an `IMessageBox` class for displaying messages in a message box.

You construct objects of the `IMessageBox` class by using an object of a class derived from `IWindow`. The `IWindow` object becomes the owner of the new message box, as follows:

```
IMessageBox messageBox(owner);
```

The following example shows you how to create a message box:

```

// Message box example
#include <iiframe.hpp>
#include <imgsgbox.hpp>
int about();
int main()
{
    about();
    return 0;
}
int about()
{
    // Create instance of IMessageBox
    IMessageBox msg(IWindow::desktopWindow());
    // Create a title for IMessageBox
    msg.setTitle("Basic MsgBox");
    msg.show("Error MsgBox",
        IMessageBox::abortRetryIgnoreButton |
        IMessageBox::defButton1 |

```

```

        IMessageBox::errorIcon |
        IMessageBox::moveable);
    return true;
}

```

The following image from Windows shows a message box created by the preceding example:



Creating a Message Window

The class `IMessageWindow` supports message IDs on a message box. `IMessageWindow` is a customizable window for displaying simple to complex messages in your application. An `IMessageWindow` can be made to look similar to an `IMessageBox` using predefined settings, or can be customized by replacing or adding to components on the message window. The default message window contains an image in the upper left, message text in the center, a button area along the bottom, and message ID text in the lower right. Through various mechanisms, you can customize or replace any of these areas on the message window. In contrast, `IMessageBox` does not provide the message ID text area and is not customizable. The appearance of the message window is platform independent, so it may look slightly different than the native controls supported by `IMessageBox`.

You can use `IMessageWindow` without customization. There are five predefined message types in the `EMessageType` enumeration that you can pass on the `IMessageWindow` constructor that give the message window a certain image and certain buttons. Using one of these predefined types gives you a message window that looks similar to an `IMessageBox`. However, `IMessageWindow` only provides the most commonly used combinations for this simple use.

In addition, there are degrees of customization you can use with `IMessageWindow`. The simplest customization is to create the message window with the `kUserDefinedMessage` type, which indicates no image and no buttons, and add your own image and buttons using `setMessageImage` and `addButton`, respectively. With these functions you can specify a predefined image or button, or create your own, and add them to the message window. This allows you to customize without deriving from `IMessageWindow`. Any buttons you create and add to the message window should be created with the parent window returned by `buttonParent`. If a button is added without this parent, an exception is thrown. The buttons are added left to right in the order that you call `addButton`. You can call `addButton` even if you used a message type other than `kUserDefinedMessage` on the constructor, and any buttons you add are placed to the right of the existing buttons. There is no limit on the number of buttons you can add.

You can obtain a pointer to a predefined button you have added to the message window by calling the `button` function and passing one of the `EButton` enumerations. You can use the pointer returned to manipulate the appearance of the button, for example, by changing the text. One reason you may want to change the default buttons or add your own buttons to the message window is so you can

provide translated text for them. The predefined button text is not translated into all languages on whose platforms you may want to run your application.

Set the styles of buttons you add or change to obtain the desired behavior for your message window. If you want a button to be the default, you must give it the `IPushButton::defaultButton` style. Likewise, if you want a help button, you must give it the `IPushButton::help` and `IButton::noPointerFocus` styles.

For further customization, you can derive from `IMessageWindow` and override one or more functions to provide your own controls to be placed in the message window. The message window allows you to provide controls for the image, the message text, and the message ID by overriding `createMessageImage()`, `createMessageText()`, and `createMessageIdText()`, respectively. You can provide any `IBitmapControl` or derived class for the image, and any `ITextControl` or derived class for the message text and message ID. The default image is an `IIconControl`, and the default message text and message ID are `IStaticText` controls. Any control you replace is positioned in the same place in the message window client area. However, you define the characteristics of the control such as minimum size, image, text, scrollability, or color.

Note that you can still add or replace buttons if you derive from `IMessageWindow` by using `IMessageWindow::addButton()`.

The most comprehensive way to customize `IMessageWindow` is to derive your own class and replace the entire client area by overriding `createClient`. You must use the `deferClientCreation` style for your override of `createClient` to be called. The function is called when the message window is shown. Whatever you return from this function is placed in the client area with no changes. You can obtain the default image, message text, and message ID controls to place in your client area by calling `createMessageImage()`, `createMessageText()`, or `createMessageIdText()`. The characteristics of the controls returned from these functions are dependent on the message type indicated in the constructor and previous calls to `setMessageText()`, `setMessageImage()`, and `setMessageId()`. To determine which buttons to place in your client area or other characteristics you may want to add, you can call `messageType()` to obtain the message type specified on the constructor.

Since `IMessageWindow` derives from `IFrameWindow`, you can use inherited functions and styles to manage the message window. Use `IFrameWindow::show()` or `IFrameWindow::showModally()` to display your message window. Use `IFrameWindow::systemModal` to make your message window system modal. There are many other frame window functions and styles you may want to apply to your message window.

You must use an `ICommandHandler` to handle processing for any buttons you add to your message window that are not listed in the `EButton` enumeration. By default, the message window will dismiss itself when any predefined button (except help) is clicked. The IDs for the predefined buttons provided by `IMessageWindow` are listed in the `EButton` enumeration. For buttons you create and add, you determine the action to be taken in your handler. If you display the message window using `showModally()`, the ID of the button clicked is returned from the call to that function.

Provide help for your message window just as you would any other frame window.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Frame Window” on page 4

“Create an Information Area” on page 9

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Chapter 2. Styles

A *style* affects the appearance and behavior of a window. Each window class has styles that are encapsulated in style objects.

Generic styles are defined in `IWindow`. Classes derived from `IWindow` can combine their own styles with those of `IWindow`.

Each window class maintains its own default style object. You can access default style objects using the static member function `defaultStyle()` and then set it using the static member function `setDefaultStyle()`. Each window class also maintains a style object called `classDefaultStyle` that corresponds to the initial setting of `defaultStyle()`.

Most window classes provide one or more constructors that accept a style object as one parameter. You can only construct a style object from existing style objects. These style objects are only used by window constructors. The style of a window can subsequently be changed and queried using the window class member functions. Also, styles which cannot change after a window has been created, in the case of, no member function is provided to change the style.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

RELATED TASKS

“Add Styles”

“Create a Frame Window” on page 4

“Task and Samples Cross-Reference Table” on page 445

Add Styles

The following sections describe how you can use bitwise operators with style objects. For the sake of simplicity, the `IComboBox` class and its styles are used for all examples. The `IBitFlag` class provides bitwise operators that you can use with the styles of the Open Class Library just as if you were using them with numbers.

Copy Styles

The assignment operator (`=`) returns one style object that is set equal to the specified style object. The value of the resulting object is equal to the value of the operand object. For example:

```
IComboBox::Style myStyle = IComboBox::dropDownType;
```

Combine Styles

The bitwise OR (`|`) operator returns a style object that is a combination of two or more style objects. The value of the resulting object is the bitwise OR of the value of the two operand objects.

You can combine any existing style objects, such as `myStyle1` and `myStyle2` in the following example, to create yet another style object. For example:

```
IComboBox::Style  
myStyle3 = myStyle1 | myStyle2;
```

This example adds the `tabStop` style to the `myStyle` object:

```
ICombobox::Style myStyle = ICombobox::dropDownType;
myStyle |= IWindow::tabStop;
```

In many cases, you can combine styles of one class with those of another class. Here, an `ICombobox` style is combined with an `IWindow` style. The documentation for each class that has styles specifies whether other classes have compatible styles that you can use when constructing objects for those classes.

Test Styles

The bitwise AND (`&`) operator returns an unsigned long integer that identifies if there are any bits common to the operand style or attribute objects. Typically, you use this operator to test whether a bitwise OR (`|`) operator has been used to combine one style object with another. For example:

```
bool isADropDown = false;
if (myStyle1 & ICombobox::dropDownType)
    isADropDown = true;
```

Negate Styles

The bitwise NOT (`~`) operator returns a negated style object. The value of the resulting object is the bitwise NOT of the value of the operand object. For example:

```
ICombobox::Style::NegatedStyle
negatedStyle = ~myStyle;
```

This code returns an object named `negatedStyle` that negates the value of the `myStyle` object.

The precedence of the AND operator (`&`) is greater than the OR operator (`|`). You must be aware of operator precedence to avoid creating invalid styles that might not be obvious.

If you do not want to consider operator precedence, specify the styles you want instead of negating others from the default Open Class Library styles.

The following example creates an invalid style that the `IViewport` constructor will reject. This causes the following:

```
IViewport::defaultStyle()
| IViewport::alwaysHorizontalScrollBar
& ~IViewport::asNeededHorizontalScrollBar
```

to be evaluated as:

```
IViewport::defaultStyle()
| (IViewport::alwaysHorizontalScrollBar
& IViewport::asNeededHorizontalScrollBar)
```

as opposed to the following:

```
(IViewport::defaultStyle()
| IViewport::alwaysHorizontalScrollBar)
& ~IViewport::asNeededHorizontalScrollBar
```

Therefore, you must consider the order and the operator precedence when you negate a style because the Open Class Library cannot change the order in which operators are evaluated in the code statement.

Setting Window Styles

You can create a window with a specific style in the following ways:

- Create a window using a constructor that accepts the style as a parameter. The following three examples illustrate this method.

This example shows how to create an entry field control with a style that is a combination of styles from IWindow and IEntryField:

```

IEntryField entryField(ID_EF1, parent, owner,
    IRectangle(10, 10, 100, 20),
    IWindow::visible |
    IWindow::tabStop |
    IWindow::group |
    IEntryField::margin |
    IEntryField::autoScroll);

```

Alternatively, you can explicitly construct the style object and pass it as a parameter as follows:

```

IEntryField::Style efStyle = IWindow::visible |
    IWindow::tabStop |
    IWindow::group |
    IEntryField::margin |
    IEntryField::autoScroll ;
IEntryField entryField(ID_EF1, parent, owner,
    IRectangle(10, 10, 100, 20),
    efStyle);

```

You can also access the default style object using the static member function defaultStyle. This simplifies the preceding example to the following:

```

IEntryField entryField(ID_EF1, parent, owner,
    IRectangle(10, 10, 100, 20),
    IEntryField::defaultStyle() |
    IWindow::tabStop |
    IWindow::group);

```

- Use the static member function setDefaultStyle to set the default style and then construct the window. For example:

```

IEntryField::Style efStyle = IEntryField::defaultStyle() |
    IWindow::tabStop |
    IWindow::group;
IEntryField::setDefaultStyle(efStyle);
IEntryField entryField(ID_EF1, parent, owner,
    IRectangle(10, 10, 100, 20));

```

- Create a window with the default style and change it using member functions of the window. The example now becomes:

```

IEntryField entryField(ID_EF1, parent, owner,
    IRectangle(10, 10, 100, 20));
// Member function of IWindow
entryField.enableGroup();
// Member function of IWindow
entryField.enableTabStop();
// Member function of IEntryField
entryField.enableAutoScroll();

```

RELATED CONCEPTS

“Chapter 2. Styles” on page 15

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Frame Window” on page 4

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Chapter 3. Events and Event Handlers

The graphical user interfaces that the Open Class Library supports are *event-driven*. A window can process a wide variety of events from many sources. Events can include the following:

- *Command Events*: when a user selects a menu item, presses an accelerator key, or selects a push button.
- *Keyboard Events*: from individual keystrokes, such as changing the contents of an entry field.
- *Mouse Events*: when a user moves the mouse or presses and releases a mouse button.
- *Paint Events*: when a portion of a window becomes *invalidated*, so that a visible part of the window cannot accurately display itself. This can happen when the user covers and uncovers a window, or when the application itself explicitly invalidates the window.
- *Resizing Events*: when the user or application changes the size of a window.

You can change the behavior of a window by changing how it processes an event. With the Open Class Library, you do this by adding an *event handler* to the window.

Event Handlers

The Open Class Library represents events with objects derived from `IEvent`. You process events routed to a window with event handler objects. Therefore, you can add customized behavior for a window by *attaching* your own event handler to it. You attach an event handler to a window by passing the window to the handler's `handleEventsFor` function.

In addition, you can reuse your handler for other windows. For example, you can reuse a handler that verifies telephone numbers wherever an entry field accepts telephone numbers. You can also attach different handlers to a window.

The following table describes various event handler classes you can use to customize the behavior of your windows:

Class	Description
<code>ICommandHandler</code>	Processes command events, such as displaying a file dialog window when a user selects Open from a menu bar.
<code>IKeyboardHandler</code>	Processes the individual keystrokes from a user, such as verifying a telephone number entered into an entry field. You would not use this class for keys that relate to command processing, such as accelerator keys, or keys that the operating system itself uses, such as the F1 key to request help.
<code>IMouseHandler</code>	Adds additional mouse functionality that the operating system does not provide.
<code>IPaintHandler</code>	Specializes the processing of a paint event, such as painting your client window with simple 2D graphics.

Class	Description
IResizeHandler	Customizes the behavior of your window when you resize your window. For example, you can reposition and resize the components of a window when it is resized, or change the size of the font it uses.

Processing Events

Each handler class has one or more virtual functions that process events. To use an event handler, you create a class derived from a handler class, then override the virtual functions to implement the behavior you want.

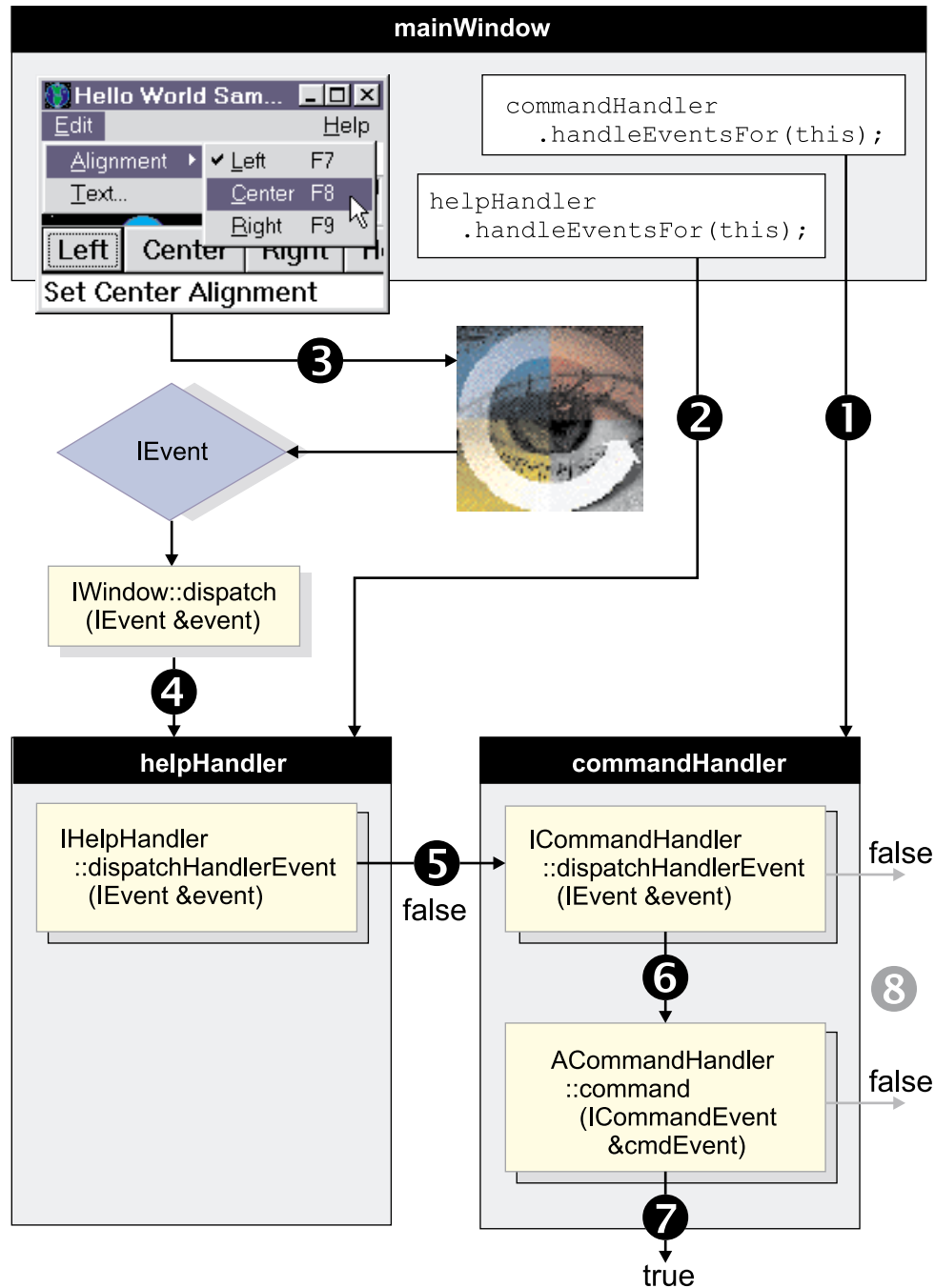
Your virtual function should either return a value of false to indicate that it has not processed the event, or true to indicate that no additional processing needs to occur for the event. Returning false allows other handlers attached to the window to be called to process the event. Handlers are called in the reverse order in which they are attached. Therefore the order in which you attach handlers can cause a handler to not receive events.

If none of the handlers attached to the window process the event, the window passes the event on for default processing by either Open Class Library or the presentation system. This could cause the event to be passed to the owner window for it to process.

All handler classes contain a dispatchHandlerEvent function to determine whether the handler needs to process an event, or return it unprocessed. If an event needs processing, the dispatchHandlerEvent function creates the appropriate event object and calls the appropriate virtual function to process the event.

The IEvent class acts as the base class for more specialized event classes. It provides general member functions to extract the message ID and message parameters. The IEvent derived classes, such as ICommandEvent, generally add more specialized functions for extracting information specific to that type of event. The IEvent class also provides a member function, setResult, for those events that require a value to be returned.

The following diagram describes how the Hello World version 5 sample handles the event that occurs when a user selects the **Center** command from the menu bar:



The main window class, `mainWindow`, handles two kinds of events:

- Command events that result from the user selecting the **Left**, **Center**, **Right**, or **Text** menu items
- Help events that result from the user selecting the **General help**, **Keys help** or **Help index** menu items

A class derived from `ICommandHandler` called `ACommandHandler` handles the command events. A class derived from `IHelpHandler` called `AHelpHandler` handles the help events. The following describes the above diagram in further detail:

1. The mainWindow constructor creates an ACommandHandler object called commandHandler. The mainWindow constructor attaches this command handler by calling the handleEventsFor function:

```
commandHandler.handleEventsFor(this);
```

2. The mainWindow constructor creates and attaches an AHelpHandler object called helpHandler:

```
helpHandler.handleEventsFor(this);
```

3. The user selects the **Center** menu item. The ID of this command is MI_CENTER. This ID has been defined in the ahellow5.h header file:

```
#define MI_ALIGNMENT    0x1500
#define MI_CENTER      0x1501
#define MI_LEFT        0x1502
#define MI_RIGHT       0x1503
#define MI_EDIT        0x1504
#define MI_TEXT        0x1505
#define MI_HELP        0x1510
```

The ahellow5.rc file resource script file maps this ID onto the menu as follows:

```
WND_MAIN MENUEX
BEGIN
    POPUP    "&Edit"                , MI_EDIT
    BEGIN
        POPUP    "&Alignment"        , MI_ALIGNMENT
        BEGIN
            MENUITEM "&Left\tF7"      , MI_LEFT
            MENUITEM "&Center\tF8"    , MI_CENTER
            MENUITEM "&Right\tF9"     , MI_RIGHT
        END
        MENUITEM "&Text..."        , MI_TEXT
    END
    POPUP    "&Help"                , MI_HELP , MFT_RIGHTJUSTIFY
    BEGIN
        MENUITEM "&General help..." , SC_HELPEXTENDED /*MIS_SYSCOMMAND*/
        MENUITEM "&Keys help..."   , SC_HELPKEYS /*MIS_SYSCOMMAND*/
        MENUITEM "Help &index..." , SC_HELPINDEX /*MIS_SYSCOMMAND*/
    END
END
```

```
WIN
```

This extract is from a Windows resource script file.

The IBM Open Class creates an IEvent object to represent this event.

4. The IBM Open Class calls the IWindow::dispatch command so that the event handlers attached to mainWindow can process the event.
5. Since helpHandler was attached last, it gets to process the event first. The help handler processes the event object by calling its dispatchHandlerEvent function. Since the event is a command event and not a help event, the dispatchHandlerEvent function returns false. A false value indicates that it did not process the event and the next event handler should process the event.
6. The command handler commandHandler processes the event object by calling its dispatchHandlerEvent function. Since the event object is a command event, the dispatchHandlerEvent creates an ICommandEvent object from the event object. The processes this ICommandEvent object by calling its command virtual function ACommandHandler::command.
7. The Hello World version 5 sample overrides the command virtual function as follows:

```

bool
ACommandHandler :: command(ICommandEvent & cmdEvent)
{
    bool eventProcessed(true);
    switch (cmdEvent.commandId()) {
        case MI_CENTER:
            frame->setTextAlignment(AHelloWindow::center);
            break;
        case MI_LEFT:
            frame->setTextAlignment(AHelloWindow::left);
            break;
        case MI_RIGHT:
            frame->setTextAlignment(AHelloWindow::right);
            break;
        case MI_TEXT:
            frame->editText();
            break;
        default:
            eventProcessed=false;
    }
    return(eventProcessed);
}

```

To obtain the ID of the command event you call the `ICommandEvent::commandId` function. The command virtual function indicates that it has processed the event by returning true. If the command virtual function returns true, no other processing occurs for the event.

8. If a handler's `dispatchEventsFor` function or its associated virtual function (such as `ICommandHandler::command`) returns false, the event object is passed to the `dispatchHandlerEvent` function of the next handler attached to the window. If no handlers process the event, the event object is passed to the window's `defaultProcedure` function for processing.

RELATED CONCEPTS

"Chapter 1. Windows" on page 1
 "Event and Event Handler Mechanism"

RELATED TASKS

"Create a Frame Window" on page 4
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

Event and Event Handler Mechanism

Encapsulation of User Interface Architectures

The Open Class Library uses events and event handlers to encapsulate and hide the message architectures of Windows, OS/2 Presentation Manager[®] (PM) and Motif in an object-oriented way.

You can either post or send a message. When a message is sent (`sendEvent`), the sender is blocked until the receiver processes the message and returns. If the sender and receiver are on the same thread, a direct function call is made to the receiver. If the sender and receiver are on different threads, the message is placed on a private area of the receiver's message queue. In either case, the sender is blocked until the receiver processes the message and returns.

When a message is posted, the message is placed on the receiver's message queue. The poster's thread then continues without waiting for a reply.

A thread switch occurs on each sent message. Continuously posting messages to another thread is less expensive than sending them.

Notifiers and Observers

The notification and observer mechanism is not an extension of the event and handler mechanism. They are designed for two different purposes. IBM Open Class uses handlers and event objects to provide a platform independent abstraction of the underlying operating system UI events and event processing.

A handler allows you to watch for certain events, perform some work in response to the event before passing it on, or even prevent it from being passed on to other handlers (including the control's window procedure). Notifications and observers are not related to the underlying operating system events and event processing. They are related to the interfaces of the IBM Open Class classes.

Multiple observers can say they are interested in knowing when certain aspects of an IBM Open Class object change or some event occurs. This is different than events and handlers in the sense that as an observer, you are notified when the aspect you are observing changes or occurs, but so are all the other observers watching that object. As an observer, you cannot stop notification from proceeding.

Events and event handling are specific to user interface objects. A non-user interface object can be a notifier and/or an observer.

The IBM Open Class objects generate notifications themselves in their member functions (for example, this occurs at the C++ object level not the presentation system control level). Since `IWindow` derives from `INotifier`, all controls know how to notify observers from any member function that does something an observer might be interested in knowing.

Valid Message IDs for Events

The Open Class Library reserves message IDs beginning at `0xFE00`. If you use the Open Class Library, define application-specific messages and events only in the range of `WM_USER` through `0xFDFF`.

WIN

`WM_USER` has a value of `0x0400`. The Windows operating system defines messages above the `WM_USER` range. Refer to your Windows documentation to ensure that you are not conflicting with these messages.

OS/2

`WM_USER` has a value of `0x1000`.

AIX

There is no `WM_USER` on Motif so you have to stay below `0xFE00`.

Tips When Using Event Handlers

When you are within a handler member function, do not delete the `IWindow` object to which the handler is attached. This will cause the Open Class Library to throw an exception.

The order in which you attach handlers can cause a handler to not receive events because handlers are called in the reverse order that they are attached.

Ensure that handlers return from virtual functions as quickly as possible to avoid locking up the system by delaying the message processing.

RELATED CONCEPTS

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Write an Event Handler”

“Extend Event Handling” on page 27

“Task and Samples Cross-Reference Table” on page 445

Write an Event Handler

In general, writing an event handler can be divided into the following steps:

1. Determine which handler class processes the event.
2. Create a class derived from that handler class and override the event-handling functions.
3. Create an instance of your derived class.
4. Attach event handlers to the window.
5. Stop handling events for the window.

The Hello World application has several event handlers. The following example illustrates how to use the above steps to process user menu selections. The code shown is from Hello World version 3.

1. Determine which handler class processes the event.

When you select a menu item, an `ICommandEvent` is generated. The handler class for this type of event is `ICommandHandler`.

2. Create a class derived from that handler class and override the event-handling functions.

The Hello World application creates a new class called `ACommandHandler` that is derived from the `ICommandHandler` class. The virtual function, `ICommandHandler::command` processes command events. The class `ACommandHandler` overrides this function to provide its own command event handling.

The following sample, taken from the `ahello3.hpp`, file, shows the class declaration of `ACommandHandler`:

```
class ACommandHandler : public ICommandHandler
{
public:
    ACommandHandler(AHelloWindow *helloFrame);
    virtual
    ~ACommandHandler() { };
protected:
    virtual bool
    command(ICommandEvent& cmdEvent);
private:
    AHelloWindow
    *frame;
};
```

The public constructor and private data member `frame` save a pointer to the frame window for which commands will be processed.

The `ACommandHandler` `command` function provides command processing for `AHelloWindow` class objects. The definition of the `command` function is taken

from `ahellow3.cpp`. The ID of the menu item is extracted from the command event object using the `commandId` member function, as follows:

```
bool ACommandHandler :: command(ICommandEvent & cmdEvent)
{
    bool eventProcessed(true);
    switch (cmdEvent.commandId())
    {
        case MI_CENTER:
            frame->setTextAlignment(AHelloWindow::center);
            break;
        case MI_LEFT:
            frame->setTextAlignment(AHelloWindow::left);
            break;
        case MI_RIGHT:
            frame->setTextAlignment(AHelloWindow::right);
            break;
        default:
            eventProcessed=false;
    }
    return(eventProcessed);
}
```

3. Create an instance of your derived class.

Define a data member from your new handler class in your application window. The following code comes from the `ahellow3.hpp` file:

```
ACommandHandler commandHandler;
```

Construct the `ACommandHandler` object in the initializer of the constructor for the application window. This is shown in the `ahellow3.cpp` file:

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
: IFrameWindow(IFrameWindow::defaultStyle() |
               IFrameWindow::minimizedIcon,
               windowId)
,menuBar(WND_MAIN, this)
,statusLine(WND_STATUS, this, this)
,hello(WND_HELLO, this, this)
,infoArea(this)
,commandHandler(this)
```

4. Attach event handlers to the window.

The base class `IHandler` provides a member function `handleEventsFor` to attach a handler to a window. In the Hello World application, `ahellow3.cpp`, the `ACommandHandler` begins processing command events for the `AHelloWindow` in its constructor with the following statement:

```
commandHandler.handleEventsFor(this);
```

5. Stop handling events for the window.

The base class `IHandler` provides a member function `stopHandlingEventsFor` to stop event processing for the window. In the Hello World application, `ahellow3.cpp`, the `ACommandHandler` stops processing command events for the `AHelloWindow` in its destructor with the following statement:

```
commandHandler.stopHandlingEventsFor(this);
```

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Hello World Version 3: Add Command Handlers and Menu Bars” on page 412

“Task and Samples Cross-Reference Table” on page 445

Extend Event Handling

Write Handlers

The IBM Open Class provides handler classes for common operating system messages. However, you may need to process messages for which there are no predefined handler classes. The `IHandler` class is the base class for handlers. All event handlers are derived from this class.

The Create Your Own Handler Sample shows a way to provide a new handler class derived from `IHandler`. The sample uses timer functions to implement a timer event handler.

To write your own handler class you generally follow these steps:

1. Derive your class from `IHandler`.
2. Optionally, override the `handleEventsFor` member function that you inherit from `IHandler`.
3. Override your `dispatchHandlerEvent` function you inherit from `IHandler` to call an event processing function.
4. Define the event processing function of your handler class.
5. Optionally, override the `stopHandlingEventsFor` function you inherit from `IHandler`.

System functions vary by operating system. If you need to call system-specific APIs you can code them within `#ifdef` preprocessor statements. This technique allows you to implement a function in different ways on different platforms within the same source file. See the topic [Create Cross-Platform Applications](#) for more information about using `#ifdef` statements.

You can find the Create Your Own Handler Sample in the `samples/ioc/ownhdr` directory:

1. Derive your class from `IHandler`.

The sample defines an `ATimeHandler` class that derives from `IHandler`. The class defines a virtual function `tick` that will process timer events received at one second intervals. The following code is from the `timehdr.hpp` file:

```
class ATimeHandler : public IHandler
{
    typedef IHandler
        Inherited;
    public:
        ATimeHandler() : timerId(0) { }
        virtual ATimeHandler() { }
        virtual ATimeHandler
            &handleEventsFor(IWindow *window),
            &stopHandlingEventsFor(IWindow *window );
    protected:
        virtual bool
            dispatchHandlerEvent(IEvent& event);
        virtual bool
            tick(IEvent& event);
    private:
        unsigned long timerId;
};
```

The sample defines the following type definition as a convenient way to reference your base class, in this case, `IHandler`:

```
typedef IHandler Inherited;
```

2. Override the `handleEventsFor` member function that you inherit from `IHandler`.

The function `ATimeHandler::handleEventsFor` starts a system timer that measures an interval of one second. The following code is from `timehdr.cpp`:

```
ATimeHandler &ATimeHandler :: handleEventsFor(IWindow *window)
{
#ifdef IC_MOTIF
    timerId = XtAppAddTimeOut (
        XtWidgetToApplicationContext ((Widget)window->handle()),
        TIME_INTERVAL,
        (XtTimerCallbackProc) postATimeHandlerEvent,
        window);
#endif
#ifdef IC_PM
    timerId = TIMER_ID;
    WinStartTimer( IThread::current().anchorBlock(),
        window->handle(), timerId, TIME_INTERVAL);
#endif
#ifdef IC_WIN
    timerId = TIMER_ID;
    SetTimer( window->handle(), timerId, TIME_INTERVAL, NULL );
#endif
    Inherited::handleEventsFor(window);
    return (*this);
}
```

The `ATimeHandler::handleEventsFor` function calls `IHandler::handleEventsFor` to complete the starting of the handler. The sample used a type definition to replace the class name `IHandler` with the name `Inherited`.

▶ AIX

Once the timer measures its defined time interval (in this case, one second) it stops. After measuring the time interval, it calls the callback function `postATimeHandlerEvent`. You must define this callback function. See [Callback Functions](#) later on this page for more information. The callback function posts a timer event to the window and starts another timer.

▶ WIN

▶ OS/2

The Windows and OS/2 timers continue to measure a given time interval (in this case, one second) until you stop the timer. After measuring the time interval, the timer posts a `WM_TIMER` event to the window you specified when you started the timer.

3. Override your `dispatchHandlerEvent` function you inherited from `IHandler`.

The `ATimeHandler::dispatchHandlerEvent` checks to see if the `IEvent` object is a timer event; it checks whether the ID of the `IEvent` object is `WM_TIMER`. If the `IEvent` object is a timer event, it calls the `tick` function to process the event. The following code is from `timehdr.cpp`:

```
bool ATimeHandler :: dispatchHandlerEvent(IEvent& event)
{
    bool eventProcessed(false);
    if ((event.eventId() == WM_TIMER) && (event.parameter1() == timerId))
    {
#ifdef IC_MOTIF
        timerId = event.parameter2();
#endif
        eventProcessed = tick(event);
    }
    return (eventProcessed);
}
```


4. Define the event processing function of your handler class.

Normally, the event processing function of a general handler class does nothing but return false. You can then create specialized classes derived from the general class where you can code implementations appropriate for specific windows. The following function, in the `timehdr.cpp` file, is the default timer tick handling function that inheriting classes should override:

```
bool ATimeHandler :: tick(IEvent& event)
{
    return (false);
}
```

The sample derives a class from `ATimeHandler` called `MyTimeHandler`. The sample then overrides the `tick` function to set the contents of a static text control to the current time. The following code is from `ownhdr.cpp`:

```
bool
MyTimeHandler::tick(IEvent& event)
{
    pText->setText( ITime().asString() );
    return false;
}
```

5. Override the `stopHandlingEventsFor` function you inherited from `IHandler`.

The `ATimeHandler::stopHandlingEventsFor` function frees all timer resources by stopping or removing the timer, depending on the platform. The following code comes from the `timehdr.cpp` file:

```
ATimeHandler
&ATimeHandler::stopHandlingEventsFor( IWindow *window )
{
    #ifdef IC_MOTIF
        XtRemoveTimeOut (timerId);
        timerId = 0;
    #endif
    #ifdef IC_PM
        if (window->isValid())
            WinStopTimer(IThread::current().anchorBlock(),
                window->handle(), timerId);
    #endif
    #ifdef IC_WIN
        if (window->isValid())
            KillTimer( window->handle(), timerId );
    #endif
    Inherited::stopHandlingEventsFor(window);
    return (*this);
}
```

Refer to the Create Your Own Handler sample application to see how to derive from `ATimeHandler` to provide a ticking clock. You can find this sample in the `samples/ioc/ownhdr` directory.

The following image, from Windows, was generated from this sample:



Callback Functions

AIX

The system timer function on AIX, `XtAppAddTimeOut`, stops after it measures its specified time interval. After it measures the time interval it calls a callback function. The sample uses the callback function to create new timers so that the sample can measure seconds on a continual basis. The following is from the `ATimeHandler::handleEventsFor` function that was shown earlier:

```
ATimeHandler &ATimeHandler :: handleEventsFor(IWindow *window)
{
#ifdef IC_MOTIF
    timerId = XtAppAddTimeOut (
        XtWidgetToApplicationContext ((Widget>window->handle()),
        TIME_INTERVAL,
        (XtTimerCallbackProc) postATimeHandlerEvent,
        window);
#endif
    // ...
    Inherited::handleEventsFor(window);
    return (*this);
}
```

The `XtAppAddTimeOut` call takes the callback function, `postATimeHandlerEvent`, as one of its arguments. The return value of `XtAppAddTimeOut` identifies that timer that it created.

The sample uses the callback function to post a timer event. The following definition of `postATimeHandlerEvent` is from the `timehdr.cpp` file:

```
#ifdef IC_MOTIF
extern void _System postATimeHandlerEvent (IWindow * window,
                                           XtIntervalId *timerUp)
{
    if (window->isValid())
    {
        IEventParameter2 newTimer = XtAppAddTimeOut (
            XtWidgetToApplicationContext((Widget>window->handle()),
            TIME_INTERVAL,
            (XtTimerCallbackProc)postATimeHandlerEvent,
            window);
        window->postEvent (WM_TIMER, IEventParameter1(*timerUp), newTimer);
    }
}
#endif
```

The call to `XtAppAddTimeOut` is the same as the one in the `ATimeHandler::handleEventsFor` function.

You can store data in `IEvent` objects. An `IEvent` object can hold two parameters of type `IEventData`. To access these parameters you call `IEvent::parameter1` and `IEvent::parameter2`. To create objects of these types use the constructors of `IEventParameter1` and `IEventParameter2`, respectively.

The above call to `IWindow::postEvent` implicitly creates an event object with two parameters: the timer that has just expired (`timerUp`) and the new timer (`newTimer`) created by `XtAppAddTimeOut`. This event object will be processed by the `ATimeHandler::dispatchHandlerEvent` function. The definition of this function is from `timehdr.cpp`:

```
bool ATimeHandler :: dispatchHandlerEvent(IEvent& event)
{
    bool eventProcessed(false);
    if ((event.eventId() == WM_TIMER) && (event.parameter1() == timerId))
    {
#ifdef IC_MOTIF
        timerId = event.parameter2();
#endif
        eventProcessed = tick(event);
    }
    return (eventProcessed);
}
```

The `ATimeHandler::dispatchHandlerEvent` first checks to see whether the event object is a timer event. If it is, it checks to see if the first parameter of the event object is equal to `timerId`. This second check determines if the event object represents the event posted right after the last timer expired. If this second check is true, it sets `timerId` to the value of the event object's second parameter (the timer that has been newly created). This ensures that the `ATimeHandler::dispatchHandlerEvent` function will process the timer event that the callback function posts after the new timer expires.

Add Event Classes

To prevent `ATimeHandler` users from having to understand how information is encoded in the two message parameters inside the event, derive an event class from `IEvent` to encapsulate this information. The following statements show an example of how to do this:

```
class ATimerEvent : public IEvent
{
public:
    ATimerEvent(IEvent &evt) : IEvent(evt) {}
    unsigned long timerNumber() const
    {
        return parameter1().asUnsignedLong();
    }
#ifdef IC_MOTIF
    unsigned long nextTimerNumber() const
    {
        return parameter2().asUnsignedLong();
    }
#endif
};
```

You can only construct objects of this class from an instance of `IEvent`. Because of the small amount of code required, the example defines the code inline.

To use the new class, change the `dispatchHandlerEvent` member function to create an instance of `ATimerEvent`. Pass this event object to the tick function. Also, change the `ATimeHandler::tick` member function to accept an `ATimerEvent` object as a parameter:

```
bool ATimeHandler :: dispatchHandlerEvent(IEvent& event)
{
    // Assume event will not be processed
    bool eventProcessed(false);
    if ( event.eventId() == WM_TIMER )
    {
        ATimerEvent timerEvent( event );
        if ( timerId == timerEvent.timerNumber() )
        {
            // This is the timer we started.
            #ifdef IC_MOTIF
                // Save the ID for the next timer event.
                timerId = timerEvent.nextTimerNumber();
            #endif
            eventProcessed = this->tick( timerEvent );
            if ( eventProcessed )
            {
                // Copy the event result.
                event.setResult( timerEvent.result() );
            }
        }
    }
    return (eventProcessed);
}
bool ATimeHandler :: tick(ATimerEvent& event)
{
    // The timer event is not processed
    return (false);
}
```

The two classes now completely encapsulate timer messages. Users of the classes do not need to know which messages are generated or how the information is encoded in the message parameters.

Restrict Window Classes from Specific Handlers

You can restrict the window classes to which a handler can be attached. The following steps show you how to restrict the attachment of the `MyTimeHandler` class to the `ITextControl` class and its derived classes.

1. Write the class declaration following this example:

```
class MyTimeHandler : public ATimeHandler
{
public:
    MyTimeHandler();
    virtual MyTimeHandler
        &handleEventsFor ( ITextControl* textWindow ),
        &stopHandlingEventsFor ( ITextControl*
            textWindow );

protected:
    virtual bool
        tick( ATimerEvent& event );

private:
    // Make these functions private
    // so that an application
    // cannot call them
    virtual MyTimeHandler
        &handleEventsFor ( IWindow* window )
    {
        return *this;
    }
}
```

```

        virtual MyTimeHandler
            &stopHandlingEventsFor ( IWindow* window )
            {
                return *this;
            }
};

```

2. Override the `handleEventsFor` member function to accept only `ITextControl` objects, as shown in the following example:

```

MyTimeHandler
    &MyTimeHandler::handleEventsFor( ITextControl* textWindow )
    {
        this->ATimeHandler::handleEventsFor( (IWindow*) textWindow );
        return *this;
    }

```

3. Override `stopHandlingEventsFor` member function to accept only `ITextControl` objects. For example:

```

MyTimeHandler
    &MyTimeHandler::stopHandlingEventsFor( ITextControl* textWindow )
    {
        this->ATimeHandler::stopHandlingEventsFor( (IWindow*) textWindow );
        return *this;
    }

```

The advantage of permitting these functions to accept only `ITextControl` objects is that from within the `tick` function you can safely cast the value returned by `IEvent::dispatchingWindow` from an `IWindow*` to an `ITextControl*`, since the handler can only be attached to `ITextControl` objects. This means that `MyTimeHandler::tick` could be implemented as follows:

```

bool MyTimeHandler::tick ( ATimerEvent& event )
{
    ITextControl
        *textControl = (ITextControl*) event.dispatchingWindow();
    textControl->setText( ITime().asString() );
    return false;
}

```

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

Create Your Own Handler Sample

Create Cross-Platform Applications

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Handle Mouse Events

Pointer devices give users the ability to perform actions directly. The Open Class Library offers classes to handle the mouse pointer.

You can use `IMouseHandler` to process a variety of mouse events. These events include button presses and releases, double-clicks, multiple button presses, and mouse moves. You can also query keyboard state information at the time a mouse event is generated.

You do not need to use `IMouseHandler` for mouse events that the operating system already supports, such as the resizing of a frame window by dragging the sizing border, or the selection of a button control by a mouse click. The following table describes some mouse events and the handler you should use instead of `IMouseHandler`:

Mouse Event	Recommended Handler
Click on a button control	<code>ICommandHandler</code>
Click on a menu bar item	<code>ICommandHandler</code>
Click on a notebook or tab control	<code>ICommandHandler</code>
Click or drag a scroll bar	The <code>IViewport</code> class provides automatic event handling for your scroll bars. If you want to customize the behavior of your scroll bars, then use <code>IScrollHandler</code>
Display a pop-up menu	<code>IMenuHandler</code>
Drag and drop (direct manipulation)	<code>IDMSourceHandler</code> , <code>IDMTargetHandler</code>

Begin by creating an `IMouseHandler` object and then attach it to any kind of window (for example, `IMultiLineEdit` or `ISetCanvas`). Although the window that the mouse is over receives a mouse event first, events are sometimes passed on for additional processing to their owner windows. A mouse event continues to travel up the owner window chain until either a handler stops it or the event is processed by the window itself. The mouse handler must return true to stop any additional processing of a mouse event.

When an `IMouseHandler` object receives a mouse event, it creates either an `IMouseEvent`, an `IMouseClickEvent`, or an `IMousePointerEvent` and routes it to a mouse handler virtual function. For the cases where mouse movement is involved (`mouseMoved`, `mouseEnter`, `mouseLeave`, `mousePointerChange`), whether and how often `IMouseHandler` calls its virtual functions is determined by the styles you pass to the `IMouseHandler` constructor. The mouse handler virtual functions are as follows:

Function	Purpose
<code>mouseClicked</code>	Processes a mouse click event.
<code>mouseEnter</code>	Processes when the mouse pointer is moved to enter a window. <code>IMouseHandler</code> does not call this function for a given window again until the mouse first moves into another window, and then back into the given window.
<code>mouseLeave</code>	Processes when the mouse pointer is moved to leave a window. <code>IMouseHandler</code> does not call this function for a given window again until the mouse first moves into the window, and then moves back out.
<code>mouseMoved</code>	Processes a mouse move event.
<code>mousePointerChange</code>	Changes the pointer when the mouse is over the handled window. If you need to change the mouse pointer for a frame window and all its children, use <code>IWindow::setMousePointer</code> .

On a two-button mouse, button1 is the left mouse button on a right-handed mouse and the right button on a left-handed mouse. Button2 is the right mouse button on a right-handed mouse and the left button on a left-handed mouse.

The `IMouseEvent` defines the following virtual functions:

Function	Purpose
<code>mousePosition</code>	Returns the position of the mouse, relative to the window handling the event.
<code>isAltKeyDown</code>	Returns true if the Alt or menu key is down when the mouse is moved.
<code>isCtrlKeyDown</code>	Returns true if the Ctrl key is down when the mouse is moved.
<code>isShiftKeyDown</code>	Returns true if the Shift key is down when the mouse is moved.
<code>windowUnderPointer</code>	Returns the handle of the window that is under the mouse pointer.

Whenever a mouse button's state changes, the `IMouseHandler` calls its `mouseClicked` function. The `IMouseClickEvent` object identifies the button, its current keyboard state, and the mouse pointer position.

The `IMouseClickEvent` defines the following virtual functions:

Function	Purpose
<code>mouseButton</code>	Returns the clicked mouse button (<code>button1</code> , <code>button2</code> , <code>button3</code>).
<code>mouseAction</code>	Returns the mouse action (clicked, double-clicked).
<code>mousePosition</code>	Returns the position of the mouse relative to the window handling the event.

`IMouseHandler` calls its `mousePointerChange` function to process a mouse pointer event when the mouse pointer enters and exits a window and/or moves within a window. The frequency that `mousePointerChange` is called is determined by the styles you use when calling the `IMouseHandler` constructor.

The `IMousePointerEvent` class defines the following functions:

Function	Purpose
<code>defaultMousePointer</code>	Returns the default mouse pointer for the window that is under the mouse. The window will use this pointer if you do not call <code>setMousePointer</code> .
<code>mousePosition</code>	Returns the position of the mouse, relative to the window handling the event.
<code>mousePointer</code>	Returns the value passed to <code>setMousePointer</code> .
<code>setMousePointer</code>	Sets the pointer to use for the window that is under the mouse.

Function	Purpose
windowId	Returns the window ID of the control that the event applies to.

RELATED CONCEPTS

- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Add Mouse Event Handling”
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

Add Mouse Event Handling

To add mouse event handling with the `IMouseHandler` class, you generally follow these steps:

1. Derive a new class from `IMouseHandler`
2. Create an object of your new mouse handling class in any window constructor
3. Call `IMouseHandler::handleEventsFor()` for each window that will recognize mouse events
4. Override any virtual function from `IMouseHandler` that you want to implement

The following code creates a split canvas as a client window. Each pane of the split canvas contains a set canvas. The set canvas located at the top of the frame window contains two bitmaps and a static text string. There is an information area at the bottom of the frame window. A mouse handler is attached to the split canvas, the set canvases, the bitmaps, and the information area. The mouse pointer changes when it is over either of the set canvases. The information area indicates when the mouse is over the set canvases or the bitmaps. The information also indicates when the mouse is clicked while over the bitmaps.

1. Derive a new class from `IMouseHandler`. The following code is from `mhsamp.hpp`:

```
class AMouseHandler : public IMouseHandler
{
public:
    AMouseHandler(MainWindow *aFrame);
protected:
    virtual bool
        mouseClicked(IMouseClickEvent & event);
    virtual bool
        mousePointerChange(IMousePointerEvent& event);
private:
    MainWindow* frame;
};
```

2. Add the new mouse handling class as a data member of the `MainWindow` class. The following code is from `mhsamp.hpp`:

```
class MainWindow : public IFrameWindow
{
public:
    MainWindow( unsigned long windowId);
    bool handleClickEvent(unsigned long id);
    bool handleChangeEvent(IMousePointerEvent& event);
private:
    ITitle          title;
```



```

ISplitCanvas      clientCanvas;
ISetCanvas        topCanvas;
ISetCanvas        bottomCanvas;
IBitmapControl   bmp1;
IBitmapControl   bmp2;
IStaticText       bmpText;
IStaticText       infoText;
IPointerHandle    ptr_bmp;
AMouseHandler     mouseHandler;
};

```

3. In the constructor of `MainWindow`, call `IMouseHandler::handleEventsFor` for each window that will recognize mouse events. The following code is from `mhsamp.cpp`:

```

MainWindow::MainWindow( unsigned long windowId)
: IFrameWindow(windowId, defaultStyle()),
  title(this, ID_TEXT),
  clientCanvas(REMOTECANVASID, this, this),
  topCanvas(TOPCANVASID, &clientCanvas, &clientCanvas),
  bottomCanvas(BOTTOMCANVASID, &clientCanvas, &clientCanvas),
  bmp1(BMP1ID, &topCanvas, &topCanvas,
        ISystemBitmapHandle::minimizeButton),
  bmp2(BMP2ID, &topCanvas, &topCanvas,
        ISystemBitmapHandle::maximizeButton),
  bmpText(BMPTXT, &topCanvas, &topCanvas,
           IRectangle(),
           IStaticText::defaultStyle() |
           IStaticText::center |
           IStaticText::top),
  infoText(INFOID, this, this, IRectangle(),
           IStaticText::defaultStyle() |
           IStaticText::top |
           IStaticText::left |
           IStaticText::wordBreak ),
  mouseHandler(this)
{
  addExtension(&infoText, IFrameWindow::belowClient,
              IFont(&infoText).maxCharHeight(),
              IFrameWindow::thickLine);
  infoText.setText(ID_TEXT);
  bmpText.setText("<=Click on bitmaps");
  clientCanvas.setOrientation(ISplitCanvas::horizontalSplit);
  topCanvas.setDeckOrientation(ISetCanvas::horizontal);
  topCanvas.add(&bmpText);
  topCanvas.add(&bmp1);
  topCanvas.add(&bmp2);
  mouseHandler.handleEventsFor(&bmp1);
  mouseHandler.handleEventsFor(&bmp2);
  mouseHandler.handleEventsFor(&topCanvas);
  mouseHandler.handleEventsFor(&bottomCanvas);
  mouseHandler.handleEventsFor(&clientCanvas);
  mouseHandler.handleEventsFor(&infoText);
  IResourceLibrary reslib;
  ptr_bmp = reslib.loadPointer(PTR_BITMAP);
  setClient(&clientCanvas);
  show();
  setFocus();
}

```

4. Override any virtual function from `IMouseHandler` that you want to implement. This example overloads two virtual functions: `mousePointerChange` and `mouseClicked`.

The `IMouseHandler` calls `mousePointerChange` whenever the user moves the mouse pointer over a handled window. The overridden version of `mousePointerChange` in this example calls `MainWindow::handleChangeEvent`.

This function changes the appearance of the mouse pointer depending on the window underneath the pointer. The following code is from `mhsamp.hpp`:

```

bool MainWindow::handleChangeEvent(IMousePointerEvent& event)
{
    unsigned long whichWindow = event.windowId();
    if ((whichWindow == BMP1ID) || (whichWindow == BMP2ID))
    {
        event.setMousePointer(
            ISystemPointerHandle(ISystemPointerHandle::kArrow));
        return true;
    }
    else if (whichWindow == BOTTOMCANVASID)
    {
        event.setMousePointer(ptr_bmp);
        infoText.setText(BOTTOMCANVAS_TEXT);
        return true;
    }
    else if (whichWindow == TOPCANVASID)
    {
        event.setMousePointer(ptr_bmp);
        infoText.setText(TOPCANVAS_TEXT);
        return true;
    }
    else
    {
        infoText.setText(ID_TEXT);
        return false;
    }
}
// ...
bool AMouseHandler::mousePointerChange(IMousePointerEvent& event)
{
    return (frame->handleChangeEvent(event));
}

```

This example calls `IMousePointerEvent::setMousePointer` to change the appearance of the pointer. The `ISystemPointerHandle` provides you with a selection of system pointers such as a text I-beam pointer and an hourglass pointer. The following function call changes the pointer to the arrow pointer:

```

event.setMousePointer(
    ISystemPointerHandle(ISystemPointerHandle::kArrow));

```

The `IMouseHandler` calls `mouseClicked` whenever the user changes the state of the a mouse button. The `mouseClicked` command takes an `IMouseClickEvent` reference as its argument. The overridden version of `mousePointerChange` determines the window underneath the mouse pointer by calling `IMouseEvent::windowUnderPointer`. It then passes the window ID of this window as an argument to `MainWindow::handleClickEvent`. This function changes the text in the information area depending on the window being clicked. The following code is from `mhsamp.cpp`:

```

bool AMouseHandler::mouseClicked(IMouseClickEvent& event)
{
    IWindow* pWindow = IWindow::windowWithHandle(event.windowUnderPointer());
    unsigned long int winId = pWindow->id();
    frame->handleClickEvent(winId);
    return true;
}
// ...
bool MainWindow::handleClickEvent(unsigned long id)
{
    switch (id)
    {
        case BMP1ID:

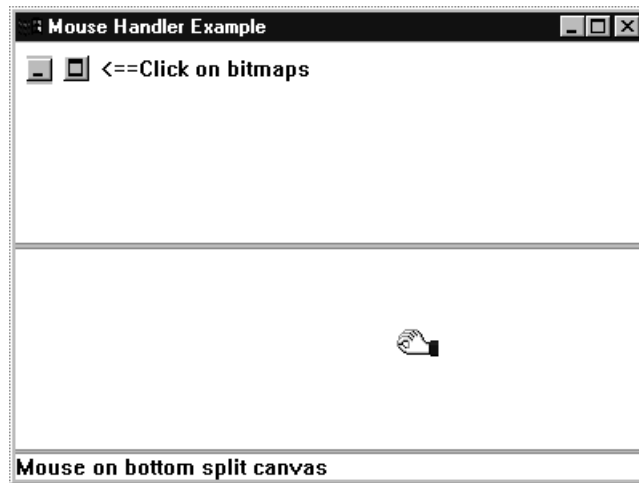
```

```

        infoText.setText(MIN_TEXT);
        break;
    case BMP2ID:
        infoText.setText(MAX_TEXT);
        break;
    }
    return true;
}

```

The following image, from Windows, shows the result of using this example:



You can find the complete source to this example in the following files:

- mhsamp.cpp
- mhsamp.h
- mhsamp.hpp
- mhsamp.rc

Derived Template Classes

Alternatively, instead of using `IMouseHandler` you can use the derived template classes `IMouseClickConnectionTo` and `IMouseMoveConnectionTo`. You can use these classes to cause a function in another class, such as a window class, to be called to process a mouse event. By using these template classes you can easily place mouse-handling code in a window class and avoid having to use multiple inheritance, friendship, or extra query and set functions to share data between classes.

The IBM Open Class has similar template classes for `ICommandHandler`, `IKeyboardHandler`, and `IPaintHandler`.

RELATED CONCEPTS

- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Handle Mouse Events” on page 33
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

"File: mhsamp.cpp"

"File: mhsamp.h" on page 42

"File: mhsamp.hpp" on page 43

"File: mhsamp.rc" on page 44

Mouse Handler Example

File: mhsamp.cpp

```
#include "mhsamp.hpp"
#include "mhsamp.h"
/*****
 * main
 * - Application entry point.
 *****/
int main()
{
    // Create our main window on the desktop
    MainWindow mainWindow(WINDOWID);
    mainWindow.sizeTo(ISize(400,300));
    // Get the current application and run it
    IApplication::current().run();
    return 0;
}
/*****
 * Class MainWindow :: MainWindow
 *****/
MainWindow::MainWindow( unsigned long windowId)
    : IFrameWindow(windowId, defaultStyle()),
      title(this, ID_TEXT),
      clientCanvas(REMOTECANVASID, this, this),
      topCanvas(TOPCANVASID, &clientCanvas, &clientCanvas),
      bottomCanvas(BOTTOMCANVASID, &clientCanvas, &clientCanvas),
      bmp1(BMP1ID, &topCanvas, &topCanvas,
           ISystemBitmapHandle::minimizeButton),
      bmp2(BMP2ID, &topCanvas, &topCanvas,
           ISystemBitmapHandle::maximizeButton),
      bmpText(BMPTXT, &topCanvas, &topCanvas,
              IRectangle(),
              IStaticText::defaultStyle() |
              IStaticText::center |
              IStaticText::top),
      infoText(INFOID, this, this, IRectangle(),
              IStaticText::defaultStyle() |
              IStaticText::top |
              IStaticText::left |
              IStaticText::wordBreak ),
      mouseHandler(this)
{
    addExtension(&infoText, IFrameWindow::belowClient,
                IFont(&infoText).maxCharHeight(),
                IFrameWindow::thickLine);
    infoText.setText(ID_TEXT);
    bmpText.setText("<==Click on bitmaps");
    clientCanvas.setOrientation(ISplitCanvas::horizontalSplit);
    topCanvas.setDeckOrientation(ISetCanvas::horizontal);
    topCanvas.add(&bmpText);
    topCanvas.add(&bmp1);
    topCanvas.add(&bmp2);
    mouseHandler.handleEventsFor(&bmp1);
    mouseHandler.handleEventsFor(&bmp2);
    mouseHandler.handleEventsFor(&topCanvas);
    mouseHandler.handleEventsFor(&bottomCanvas);
}
```

```

        mouseHandler.handleEventsFor(&clientCanvas);
        mouseHandler.handleEventsFor(&infoText);
        IResourceLibrary reslib;
        ptr_bmp = reslib.loadPointer(PTR_BITMAP);
        setClient(&clientCanvas);
        show();
        setFocus();
    }
/*****
 * Class MainWindow :: handleChangeEvent
 *****/
bool MainWindow::handleChangeEvent(IMousePointerEvent& event)
{
    unsigned long whichWindow = event.windowId();
    if ((whichWindow == BMP1ID) || (whichWindow == BMP2ID))
    {
        event.setMousePointer(
            ISystemPointerHandle(ISystemPointerHandle::kText));
        return true;
    }
    else if (whichWindow == BOTTOMCANVASID)
    {
        event.setMousePointer(ptr_bmp);
        infoText.setText(BOTTOMCANVAS_TEXT);
        return true;
    }
    else if (whichWindow == TOPCANVASID)
    {
        event.setMousePointer(ptr_bmp);
        infoText.setText(TOPCANVAS_TEXT);
        return true;
    }
    else
    {
        infoText.setText(ID_TEXT);
        return false;
    }
}
/*****
 * Class AMouseHandler :: mouseClicked
 *****/
bool AMouseHandler :: mouseClicked (IMouseClickEvent & event)
{
    IWindow * pWindow =IWindow::windowWithHandle(event.windowUnderPointer());
    unsigned long int winId = pWindow->id();
    frame->handleClickEvent(winId);
    return false;
}
/*****
 * Class MainWindow :: handleClickEvent
 *****/
bool MainWindow::handleClickEvent(unsigned long id)
{
    switch (id)
    {
        case BMP1ID:
            infoText.setText(MIN_TEXT);
            break;
        case BMP2ID:
            infoText.setText(MAX_TEXT);
            break;
    }
    return true;
}
/*****
 * Class AMouseHandler :: AMouseHandler
 *****/

```

```

AMouseHandler :: AMouseHandler(MainWindow *aFrame)
{
    frame=aFrame;
}
/*****
* Class AMouseHandler :: mousePointerChange *
*****/
bool AMouseHandler :: mousePointerChange (IMousePointerEvent& event)
{
    return (frame->handleChangeEvent(event));
}

```

RELATED CONCEPTS

“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Add Mouse Event Handling” on page 36
“Handle Mouse Events” on page 33
“File: popmenu.cpp” on page 69“File: mhsamp.h”
“File: mhsamp.hpp” on page 43
“File: mhsamp.rc” on page 44
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

File: mhsamp.h

```

#define WINDOWID      1001
#define BMPTXT       1002
#define REMOTECANVASID 1003
#define INFOID       1004
#define TOPCANVASID  1005
#define BOTTOMCANVASID 1006
#define BMP1ID       2001
#define BMP2ID       2002
#define PTR_BITMAP   2003
#define BITMAP_TEXT  3001
#define ID_TEXT      3002
#define TOPCANVAS_TEXT 3003
#define BOTTOMCANVAS_TEXT 3004
#define MAX_TEXT     3005
#define MIN_TEXT     3006

```

RELATED CONCEPTS

“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Add Mouse Event Handling” on page 36
“Handle Mouse Events” on page 33
“File: popmenu.cpp” on page 69“File: mhsamp.cpp” on page 40
“File: mhsamp.hpp” on page 43
“File: mhsamp.rc” on page 44
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

File: mhsamp.hpp

```
#include "mhsamp.h"
#include <imoushdr.hpp>
#include <ibmpctl.hpp>
#include <iframe.hpp>
#include <isplitcv.hpp>
#include <isetcv.hpp>
#include <istattxt.hpp>
#include <ititle.hpp>
#include <ifont.hpp>
#include <ihandle.hpp>
class MainWindow;
/*****
 * Class AMouseHandler
 *****/
class AMouseHandler : public IMouseHandler
{
public:
    AMouseHandler(MainWindow *aFrame);
protected:
    virtual bool mouseClicked (IMouseClickEvent & event);
    virtual bool mousePointerChange(IMousePointerEvent& event);
private:
    MainWindow * frame;
};
/*****
 * Class MainWindow
 *****/
class MainWindow : public IFrameWindow
{
public:
    MainWindow( unsigned long windowId);
    bool handleClickEvent(unsigned long id);
    bool handleChangeEvent(IMousePointerEvent& event);
private:
    ITitle          title;
    ISplitCanvas    clientCanvas;
    ISetCanvas      topCanvas;
    ISetCanvas      bottomCanvas;
    IBitmapControl  bmp1;
    IBitmapControl  bmp2;
    IStaticText     bmpText;
    IStaticText     infoText;
    IPointerHandle  ptr_bmp;
    AMouseHandler   mouseHandler;
};
```

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Add Mouse Event Handling” on page 36

“Handle Mouse Events” on page 33

“File: popmenu.cpp” on page 69“File: mhsamp.cpp” on page 40

“File: mhsamp.h” on page 42

“File: mhsamp.rc” on page 44

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: mhsamp.rc

WIN

This is a Windows resource script file.

```
#include "mhsamp.h"
PTR_BITMAP CURSOR hwe.cur
STRINGTABLE
BEGIN
    BITMAP_TEXT,      "<=Click on bitmaps"
    ID_TEXT,          "Mouse Handler Example"
    TOPCANVAS_TEXT,  "Mouse on top split canvas"
    BOTTOMCANVAS_TEXT, "Mouse on bottom split canvas"
    MAX_TEXT,        "Maximize bitmap clicked"
    MIN_TEXT,        "Minimize bitmap clicked"
END
```

AIX

OS/2

This is an AIX or OS/2 resource script file:

```
#include "mhsamp.h"
POINTER PTR_BITMAP hwe.cur
STRINGTABLE
BEGIN
    BITMAP_TEXT,      "<=Click on bitmaps"
    ID_TEXT,          "Mouse Handler Example"
    TOPCANVAS_TEXT,  "Mouse on top split canvas"
    BOTTOMCANVAS_TEXT, "Mouse on bottom split canvas"
    MAX_TEXT,        "Maximize bitmap clicked"
    MIN_TEXT,        "Minimize bitmap clicked"
END
```

Additional Files



Cursor: hwe.cur

RELATED CONCEPTS

“Chapter 3. Events and Event Handlers” on page 19

“Chapter 21. Resources” on page 297

RELATED TASKS

“Add Mouse Event Handling” on page 36

“Handle Mouse Events” on page 33

“Convert Application Resources” on page 395

“File: popmenu.cpp” on page 69 “File: mhsamp.cpp” on page 40

“File: mhsamp.h” on page 42

“File: mhsamp.hpp” on page 43

Set Time Intervals

A timer controls when certain events will occur based on a specified timed basis. You can use the `ITimer` class objects to set time-interval-based operations. The Open Class Library uses timers in fly-over help and with `IAnimatedButton`. The Open Class Library provides the following classes to create timer objects:

Class	Description
ITimer	This class creates objects representing operating system timers. An object of this class calls a specified function in your class when the timer expires. The timer continues to call the function each time the specified time limit expires.
ITimerFn	An abstract timer function class. An instance of this class is passed into ITimer when you start it. You can then delete the ITimer object, and the timer will continue to run.
ITimerMemberFn	Template class for the timer member function. This class inherits from ITimerFn.
ITimerMemberFn0	Another template class for the timer member function. This class does not accept any parameters.

Create Timers

Use the ITimer class objects to define and set time-interval-based operations for your current program. You can also use ITimer objects to create additional time-interval-based operations by instantiating new ITimer objects and starting them. You start a timer using the ITimer::start() member function.

The following is an example of starting a new timer to execute a member function:

```
ITimer timer;
timer.start(new ITimerMemberFn<MyClass>(*this, foo));
```

You construct instances of this class in one of the following ways:

- Use the default constructor. Use this to build an ITimer that is started using the ITimer::start() member function.
- Reference the code that you want to execute with the timer. This form of the constructor is used to construct a new ITimer and immediately start it. It is equivalent to constructing using the default constructor and then using ITimer::start(). You can specify a time-interval to be used (in thousandths of a second) with this constructor. A time-interval of one second is the default.
- Use the timer ID of a currently started ITimer object.

ITimer has a virtual destructor that deallocates resources. This destructor, however, does not stop the timer. The timer continues to run until you call ITimer::stop(). The reference to the class that is passed into the constructor must be dynamically allocated. Otherwise, it results in an exception when the timer is stopped. Furthermore, stopping the timer deletes the allocation of the object making any previous pointers to it illegal.

Timer identifiers are limited to the value 0x7FFF to allow portability to OS/2 where this limit is imposed by the operating system.

The ITimerFn class is an abstract timer function class. Objects of this class represent events dispatched when an active timer expires. These objects are reference-counted to manage their destruction after the timer has stopped. This reference to an ITimerFn is then passed to ITimer::start(). The timerExpired() function is called when the timer expires.

The `ITimerMemberFn` template class is an `ITimerFn`-derived class for dispatching C++ member functions against an object when a timer expires. You can use instances of this class to dispatch C++ member functions when a timer expires. The template argument is the class of the object for which the member functions are called. The constructor for such objects requires two arguments:

- A reference to the object the member function is to be applied to.
- A pointer to the member function. This member function must return a void and accept a timer reference argument.

This class overrides the inherited function `timerExpired()`.

To create a timer, do the following:

1. Write a function for what you want to do on a timed basis.
2. Create an `ITimerMemberFn` or `ITimerMemberFn0` object. For a function that will be repeating an action on a timed basis, use `ITimerMemberFn0`. Use `ITimerMemberFn` if you need to know the timer ID. Use `ITimerMemberFn0` if you do not need the timer ID.

The `ITimerMemberFn` constructor takes a pointer to a member function as an argument. This member function must return a void parameter and accept an unsigned long formal parameter that represents the timer ID. The timer ID is passed to you when the timer stops. For `ITimerMemberFn0`, you define a function that takes no parameters and you do not get the timer ID.

An example would be if you wanted a window to refresh itself every 30 minutes. If you wanted a window displayed for five seconds and then have it disappear, you would use `ITimerMemberFn` because the action only occurs once. You would then use `ITimer::stop()` to delete the timer.

If you call `setInterval()` with any value, the timer is stopped and restarted with the new interval. Restarting the timer with a value of zero causes the timer to timeout immediately and results in a call to your timer member function. If you call `ITimer::start()` without stopping the timer, you will get an `InvalidRequest` exception.

The following is an example of a timer created to run the public member job in `MyClass`:

```
class MyClass
{
public:
    void job(unsigned long timerId)
    {
        // Code to run.
    }
};
//...
MyClass myObj;
ITimer timer;
timer.start(new ITimerMemberFn<MyClass>(myObj, MyClass::job));
```

In the preceding example, we used `ITimerMemberFn` because the job member function returns void and accepts an unsigned long parameter, `timerId`. When parameters are not accepted by the member function, you can use `ITimerMemberFn0`.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Write an Event Handler” on page 25

“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443

“Task and Samples Cross-Reference Table” on page 445

Add Time Intervals

You can specify an event to occur at regular intervals by following these steps:

1. Define a class that derives from `ITimerFn`.
2. Create a `ITimer` object.
3. Define the interval for the timer.
4. Override the `timerExpired()` virtual function. The application calls this function when the timer expires.
5. Create a function that turns the timer on and off.

The Hello World Version 6 sample uses the `ITimerFn` class to make stars twinkle. The `twinkleStars()` function redraws each of the stars at a different intensity. The application calls this function at regular intervals to “twinkle” the stars:

1. Define a class that derives from `ITimerFn`. The `ITimerFn` class has a virtual function called `timerExpired()`. The application calls this function when the timer stops. The following code is from `aearthw6.hpp`:

```
class ATwinkleTimeHandler : public ITimerFn
{
public:
    // Constructor
    // - constructs the object with a pointer to the
    //   earth window
    ATwinkleTimeHandler (AEarthWindow *a) : aew(a) {}
    // Override timerExpired function
    // - the timerExpired() function is called each time
    //   the ITimer expires.
    virtual void
        timerExpired (unsigned long);
private:
    AEarthWindow
        *aew;
};
```

2. Create an `ITimer` object. The following code is from `aearthw6.hpp`:

```
class AEarthWindow : public IDrawingCanvas
{
public:
    AEarthWindow(unsigned long windowId,
                 IWindow * parentownerWindow,
                 const IRectangle& rect=IRectangle());
    // ...
    bool twinkleStars();
    // ...
    virtual AEarthWindow &enableTwinkle(bool turningOn=true),
    // ...
private:
    // ...
    ITimer twinkleTimer;
    // ...
};
```

3. Define the interval for the timer. This interval determines the duration between successive calls to `timerExpired()`. The following code is from `aearthw6.cpp`:

```

// Constructor for the Earth window
AEarthWindow :: AEarthWindow(unsigned long windowId,
                               IWindow * parowWindow,
                               const IRectangle& rect)
    : // ...
{
    // ...
    twinkleTimer.setInterval(500);
    // ...
}

```

4. Override the `timerExpired()` virtual function. The overridden version of `timerExpired()` in this sample calls the member function `twinkleStars()`. The following code is from `aearthw6.cpp`:

```

// Make the stars twinkle and refresh.
bool
AEarthWindow::twinkleStars()
{
    if (twinkling)
    {
        // If the stars are twinkling, then call the flicker
        // function for each and redraw them on the
        // IDrawingCanvas.
        for (int i=0;i<13;i++)
            starList[i].flicker();
        paintWorld();
    }
    return true;
}
// ...
// Process ATimeHandler ticks by calling the AEarthWindow
// function for twinkling the stars.
void ATwinkleTimeHandler::timerExpired(unsigned long timeid)
{
    aew->twinkleStars();
}

```

5. Create a function that turns the timer on and off. This function creates a new `ITimerFn` object to start the timer. The following code is from `aearthw6.cpp`:

```

// Make the stars twinkle.
AEarthWindow
&AEarthWindow :: enableTwinkle(bool turningOn)
{
    if (turningOn && !twinkling)
    {
        twinkling=true;
        twinkleTimer.start(new ATwinkleTimeHandler(this));
        twinkleStars();
    }
    else if (!turningOn && twinkling)
    {
        twinkling=false;
        twinkleTimer.stop();
        addObjectstoGroup();
    }
    return (*this);
}

```

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Set Time Intervals” on page 44

“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Monitor Text-Change Events

► AIX

WIN

The classes `ITextChangeEvent` and `ITextChangeHandler` are not supported on OS/2.

Objects of the `ITextChangeEvent` class represent a pending change to the text data in an editable text control.

You can verify any change to the text data of these controls using the `ITextChangeHandler` class. Changes to the text might include insertions, replacements, and deletions.

The change can be via keyboard entry, cut and paste, or using the text control’s functions. Depending on how this event is processed, the change will be successful (either exactly as proposed, or you can modify the proposed text data), or unsuccessful. If successful, a control event is sent to the `IEditHandler` signaling that the change has occurred.

An `ITextChangeHandler` constructs and processes `ITextChangeEvents`. They can be constructed elsewhere as well. You must write your own handler class derived from `ITextChangeHandler`. You can then use this handler to gain access to `ITextChangeEvents`, allowing you to edit and/or verify proposed changes to an editable text control.

`ITextChangeEvents` are first dispatched to the handlers attached to the control itself, then to the handlers attached to the owner window of the control.

`ITextChangeHandler` is called after `IKeyboardHandlers`, but before `IEditHandlers` and before any of the changes in the text values show up on the screen.

You can create a handler derived from `ITextChangeHandler` and attach it to either the control whose input users can change or to the control’s owner window. You do this by calling `IHandler::handleEventsFor` to pass the appropriate control window or owner window to the text-change handler.

When the text-change handler receives a text-change event, it creates an `ITextChangeEvent` object and routes that object to the `textChange` virtual function. You must override this virtual function to supply your own specialized processing of a text-change event. In your processing, you decide whether the proposed change is successful (either exactly as proposed, or you can modify the proposed text data) or unsuccessful. The `ITextChangeEvent`, which is routed to the virtual function, contains both the proposed data and the `allowChange` flag. This is the event you modify in your processing.

When processing, you can increase or decrease the size of the proposed text or you can change the value of of the proposed text. Handlers which are called after yours

can set the `allowChange` flag to false. After all the handlers have been called, the final state of `allowChange` determines whether a control event is created. The default behavior is successful (true). The control event is sent to `IEditHandler` signaling the change to the control.

The return value from the virtual function specifies whether the control event is passed on for additional processing. If true, the text-change event requires no additional processing and is not passed to another handler.

If false, the text-change event is passed to the next handler for additional processing and if there is another handler for the control or window, the text-change event is passed to the next handler. If this is the last handler for the control, call `IWindow::dispatch` to dispatch the text-change event to the control's owner window. If this is the last handler for the owner window, call `IWindow::defaultProcedure` to process the text-change event.

Note that additional processing by another handler can alter the proposed data again, and possibly modify the `allowChange` flag.

RELATED CONCEPTS

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Use Keyboard Accelerators"

"Handle Text-Change Events" on page 52

"Write an Event Handler" on page 25

"Task and Samples Cross-Reference Table" on page 445

Use Keyboard Accelerators

A *keyboard accelerator* is a keystroke or a series of combined keys that generate a command message for an application. Using a keyboard accelerator lets a user quickly access commands. It has the same effect as choosing a menu item.

Your application can load accelerators either statically or at runtime:

- For static loading, you load the definitions of keyboard accelerators from a resource file (via an accelerator table resource).
- For dynamic or runtime loading, you create and/or modify accelerator keys programmatically.

Then, the application frame window loads the accelerator table with the same resource ID, if found. The Open Class Library also provides `IAccelerator`, `IAcceleratorKey`, and `IAcceleratorTable` classes to add accelerators at runtime.

You can also add accelerators to a window by using the `IFrameWindow::accelerator` style object. For example, the Hello World version 4 sample passes the `IFrameWindow::accelerator` style object as a parameter for the main window constructor. The following is an excerpt from `ahellow4.cpp`:

```
AHelloWindow :: AHelloWindow(const unsigned long windowId)
    :IFrameWindow(IFrameWindow::defaultStyle() |
                  IFrameWindow::minimizedIcon |
                  IFrameWindow::accelerator,
                  windowId)
```

Accelerator Tables

WIN

On Windows, both upper and lower case accelerators are treated as the same accelerator. For example, pressing “Ctrl-a” or “Ctrl-A” works with an accelerator defined in a resource file as follows:

```
WND_MAIN ACCELTABLE
BEGIN
    "A" , MI_SELECT_ALL , CONTROL
    "D" , MI_DESELECT_ALL , CONTROL
END
```

AIX

OS/2

However, OS/2 and AIX differentiates between upper and lower case accelerators. Pressing “Ctrl-a” or “Ctrl-A” requires separate accelerator entries as follows:

```
ACCELTABLE WND_MAIN
BEGIN
    "A" , MI_SELECT_ALL , CONTROL
    "a" , MI_SELECT_ALL , CONTROL
    "D" , MI_DESELECT_ALL , CONTROL
    "d" , MI_DESELECT_ALL , CONTROL
END
```

Without accelerators, a user might generate commands by pressing the Alt or F10 keys for OS/2 and Windows, or the F10 key for Motif. These keys access the menu bar where you can use the Arrow keys to select the item, and then press the Enter key to choose the item. In contrast, accelerators let users generate commands with fewer keystrokes.

Although accelerators are usually used to generate existing commands as menu items, they also can send commands that have no menu-item equivalent.

Accelerator Tables

An *accelerator table* contains an array of accelerators.

Accelerator tables exist at two levels within the operating system: a single accelerator table for the system queue and individual accelerator tables for application windows.

Accelerators in the system queue apply to all applications. For example, the F1 key always generates a help message.

Having accelerators for individual application windows ensures that an application can define its own accelerators without interfering with other applications. An accelerator for an application window can override the accelerator in the system queue.

An application can modify both its own accelerator table and the system’s accelerator table.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Handle Text-Change Events”
“Monitor Text-Change Events” on page 49
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

Handle Text-Change Events

The IEditHandler class handles events resulting from a user changing a control’s input value, such as changing the value of an entry field or moving the arm of a slider. IEditHandler objects process edit events for the following controls:

- ICircularSlider
- IComboBox
- IEntryField
- IMultiLineEdit
- INumericSpinButton
- IProgressIndicator
- ISlider
- ITextSpinButton

When the edit handler receives an edit event, it creates an IControlEvent object and routes that object to the IEditHandler::edit() virtual function. You override this virtual function to supply your own specialized processing of an edit event.

> AIX

> WIN

Objects of the ITextChangeHandler class process text change events for the following controls:

- IComboBox
- IEntryField
- IMultiLineEdit
- INumericSpinButton
- ITextSpinButton

Objects of the ITextChangeEvent class represent a pending change to the text data in an editable text control.

Text-change events describe a proposed change to the data in an editable text control. Such events result from a user attempting to change an input text value.

> OS/2

The ITextChangeHandler class is not supported for OS/2 Presentation Manager.

ITextChangeHandler handles an event that the text control generates before updating its text. IEditHandler handles an event generated after the text control has updated its text.

RELATED CONCEPTS

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Monitor Text-Change Events" on page 49

"Write an Event Handler" on page 25

"Task and Samples Cross-Reference Table" on page 445

Chapter 4. Menus

A *menu* is a list of predetermined items (like text strings, bitmaps, or images drawn by an application) from which a user can choose using a mouse or keyboard. The types of menus are the menu bar, pop-up menus, and the system menu. You can layer the choices on a menu into a hierarchy so that selecting a choice on a menu bar can display further choices in a pull-down menu. Similarly, selecting a choice in a pop-up, system, or pull-down menu can display additional choices in a cascaded menu.

You can dynamically create a menu using program functions or load it from a resource file. A menu is always owned by another window, usually a frame window.

A *menu item* is a choice on a menu or a separator between choices. A menu item can represent either a command choice, or it can represent an item that, when selected, brings up an additional list of choices to choose from. When a user selects a menu item which represents a command choice, it sends a command event to the owner of the menu. You can process this event with a command handler.

Menu Bars and Pull-Down Menus

The *menu bar* is the area near the top of a window, below the title bar and above the client area of the window. A menu bar is a horizontal menu that contains a list of choices. When a user selects a choice on a menu bar, a pull-down menu associated with that choice is displayed.

A *pull-down* menu is a vertical menu that appears below the menu bar as the result of selecting a menu item that represents this list of choices. (If there is not enough room to show the pull-down menu, it will appear above the menu bar.) Typically, all of the menu items on a menu bar represent pull-down menus instead of command choices. The pull-down menus are ordinarily hidden, but they become visible when the user makes a selection from the menu bar.

In turn, selecting a menu item in the pull-down can display more menu items in what is called a *cascaded menu*. Several GUI style guides recommend that you use no more than three levels of cascaded menus, and create a new frame window when you have more than three.

The Open Class Library provides the `IMenu`, `IMenuBar`, and `IMenuItem` classes for creating menu bars, menu items, and pull-down menus, and querying the contents of an existing menu.

Use the classes `IMenuBar`, `IPopUpMenu`, `ISubmenu`, and `ISystemMenu` to work with your menus and to add and remove items from menus. These classes derive from `IMenu`. This class provides member functions that you use to manipulate menus according to the needs of your application. You can add, replace, remove, select, and disable menu items in menus. This class also provides functions to access properties such as the following:

- The number of menu items in the menu
- The bounding rectangle for menu items

Objects of the `IMenu` (and derived) classes are not windows themselves. The `IMenu` class does not inherit from the `IWindow` class.

To support menus loaded from a resource file, IMenu classes have the following requirements:

-

AIX

OS/2

Menu resources must be defined using the MENU keyword.

-

WIN

Menu resources must be defined using the MENUEX keyword.

- Unique IDs must be provided for all pop-up menu items, indicated by the POPUP keyword.

AIX

IMenu does not support conditional cascading behavior in menus.

Pop-Up Menu

A *pop-up menu* is displayed next to a selected object when a user presses the appropriate key or mouse button. A pop-up menu contains choices that can be applied to the object when the menu is displayed.

The Open Class Library provides the IPopupMenu and IMenuHandler classes to manipulate pop-up menus. To create a pop-up menu in your application, derive your class from IMenuHandler, override the makePopupMenu function, and construct an IPopupMenu object.

You can dynamically add, remove, or change items in the pop-up menu by overriding the IMenuHandler::menuShowing virtual function and making the appropriate changes to the ISubmenu object passed to the menuShowing function.

System Menu

The system menu in the upper-left corner of a standard frame window is different from the menus defined by the application. The system menu is controlled and defined by the operating system. However, using Open Class Library, you can enable and disable items on the system menu, add or delete items, or even decide not to include the system menu on your frame window. The ISystemMenu class gives you access to the system menu and allows you to manipulate its items. You can create a frame window with a system menu by constructing an IFrameWindow object with the style IFrameWindow::systemMenu. This is a default style for IFrameWindow.

Window Menu Button

AIX

The window menu displayed by the window menu button in the upper-left corner of a standard Motif frame window is different from the menus defined by the application. The window menu button is controlled and defined almost exclusively by the system; your only decision about it is whether to include it when creating a frame window.

Menu Bar Interaction

When the user selects a menu item that represents a command choice, any command handlers attached to the frame window are notified of the command event, indicating which choice the user selected. If the command event is not

processed by the frame window, the frame window will send it to its client window for processing (if it has a client window). See `ICommandHandler` for more information about its use.

This generally applies to pop-up menus, except the command events first go to the owner window of the pop-up menu. If the window receiving a command event is derived from `ICanvas` or is an `IContainerControl` and the command event is not handled, the canvas or container will send the event to its owner window for processing.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Add Menus to Your Application”

“Add a Menu Bar” on page 59

“Create a Pop-Up Menu” on page 62

“Create a Pop-Up Menu for a List Box” on page 65

“Dynamically Modify a Menu with the `ISubmenu Class` ” on page 75

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Add Menus to Your Application

A typical application uses a menu bar with several pull-down submenus. This section shows you two ways to create a menu bar:

1. Most applications load a menu resource script to define the menus they need, and there are two ways to load this predefined menu resource.
2. Some applications dynamically create the menu bar programmatically using `IMenu` member functions.

Applications can also choose to mix and match these methods to obtain the results desired. An example of this would be to dynamically load a menu bar containing all possible menu options and then programmatically remove those items at run time which are not needed, depending on the state of the application.

`IMenu` is no longer derived from `IWindow` to more closely model the behavior on the Windows operating system. On OS/2 and Motif, you still have access to an `IWindow` object if needed. Draw-item menus are supported on all platforms.

Load a Menu From a Resource File

Version 6 of the Hello World application creates a frame window with the following statements and then loads a menu bar from the resource file. In `ahellow6.hpp` , we do the following:

```
class AHelloWindow : public IFrameWindow
{
    // ...
private:
    IMenuBar menuBar;
    // ...
};
```

In `ahellow6.cpp` , we do the following:

```

AHelloWindow::AHelloWindow(unsigned long windowId)
: IFrameWindow(IFrameWindow::defaultStyle() |
               IFrameWindow::minimizedIcon |
               IFrameWindow::accelerator,
               windowId)
, menuBar(windowId, this)
// ...

```

In the preceding example, the frame window is constructed without specifying the `IFrameWindow::menuBar` style on the constructor. During initialization of the frame, it will then construct an `IMenuBar` object and will load the menu bar from the resource file, using the `windowId` passed as the identifier for that resource. The content of the menu bar is defined in the `ahellow6.rc` resource file as in the following statements:

```

WND_MAIN MENUEX
BEGIN
  POPUP "&Edit", MI_EDIT
  BEGIN
    POPUP "&Alignment", MI_ALIGNMENT
    BEGIN
      MENUITEM "&Left\tf7", MI_LEFT
      MENUITEM "&Center\tf8", MI_CENTER
      MENUITEM "&Right\tf9", MI_RIGHT
    END
    MENUITEM "&Text...", MI_TEXT
    MENUITEM "&Font...", MI_FONT
  END
  POPUP "&Settings", MI_SETTINGS
  BEGIN
    MENUITEM "&Read from profile", MI_READSETS
    MENUITEM "&Open ...", MI_OPENSETS
    MENUITEM "&Save to profile", MI_SAVESETS
  END
END

```

WIN

This is an example of a Windows resource file.

This same menu bar can also be created and loaded in the frame window constructor by specifying the `IFrameWindow::menuBar` style. If an `IMenuBar` object is needed, it can be created by wrapping the one loaded by the frame window constructor. The following statements demonstrate this:

```

AHelloWindow :: AHelloWindow(unsigned long windowId)
: IFrameWindow(IFrameWindow :: defaultStyle() |
               IFrameWindow :: menuBar
               IFrameWindow :: minimizedIcon
               IFrameWindow :: accelerator,
               windowId)
, menuBar(this, IMenuBar::wrapper)

```

Dynamically Create a Menu Using IMenu Member Functions

You can create a menu bar dynamically at run time. This can be done using the `IMenuBar` constructor and specifying the `IMenuBar::empty` style and the frame window the menu bar is for. The menu can then be populated using `IMenu` member functions. The `addText` member function is used to add a menu item to the menu bar with the ID specified and the string passed. If a third parameter is passed, the menu item is added into the pull-down menu specified instead of the menu bar itself. The `addSubmenu` member functions are used to cause a menu item to display a pull-down menu. The following statements demonstrate building the same menu that is defined in the preceding resource file:

```

IMenuBar* menuBar = new IMenuBar(this);
(*menuBar)
    .addText(MI_EDIT, "&Edit")
    .addSubmenu(MI_EDIT)
        .addText(MI_ALIGNMENT, "&Alignment", MI_EDIT)
        .addSubmenu( MI_ALIGNMENT )
            .addText( MI_LEFT      , "&Left\tF7",   MI_ALIGNMENT )
            .addText( MI_CENTER    , "&Center\tF8", MI_ALIGNMENT )
            .addText( MI_RIGHT     , "&Right\tF9",  MI_ALIGNMENT )
        .addText( MI_TEXT        , "&Text...",  MI_EDIT )
        .addText( MI_FONT        , "&Font...",  MI_EDIT )
    .addText( MI_SETTINGS, "&Settings" )
    .addSubmenu( MI_SETTINGS )
        .addText( MI_READSETS , "&Read from profile", MI_SETTINGS )
        .addText( MI_OPENSETS , "&Open ...",          MI_SETTINGS )
        .addText( MI_SAVESETS , "&Save to profile",   MI_SETTINGS );

```

RELATED CONCEPTS

- “Chapter 4. Menus” on page 55
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Add a Menu Bar”
- “Create a Frame Window” on page 4
- “Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

Add a Menu Bar

The Hello World Version 3 contains a menu bar with only one submenu named **Alignment**. When you run the sample and select **Alignment**, the pull-down menu is displayed. The choices in the pull-down menu are **Left**, **Center**, and **Right**. When you select one of the choices, the text string in the client window aligns to the selected position and a check mark appears beside the selected item.

1. Define the menu bar. The Hello World Version 3 sample uses a resource script file to define the text for the menu bar and its associated pull-down menu.

▶ WIN

The following is from the ahellow3.rc file for Windows:

```

WND_MAIN MENUEX
BEGIN
    POPUP "&Alignment"      , MI_ALIGNMENT
    BEGIN
        MENUITEM "&Left"      , MI_LEFT
        MENUITEM "&Center"    , MI_CENTER
        MENUITEM "&Right"     , MI_RIGHT
    END
END

```

▶ AIX

▶ OS/2

The following is from the ahellow3.rc file for AIX and OS/2:

```

MENU_WND_MAIN
BEGIN
    SUBMENU "~Alignment"    , MI_ALIGNMENT
    BEGIN
        MENUITEM "~Left"    , MI_LEFT
        MENUITEM "~Center"  , MI_CENTER
        MENUITEM "~Right"   , MI_RIGHT
    END
END

```

To assist keyboard users of your application, underline the mnemonic in each menu item.

▶ WIN

Use the ampersand (&).

▶ OS/2

▶ AIX

Use the tilde (~).

2. Declare and create the menu bar. This sample declares an `IMenuBar` object in the main window class called `AHelloWindow`. The following excerpt is from `ahellow3.hpp`:

```

class AHelloWindow : public IFrameWindow
{
public:
    // ...
private:
    IMenuBar
        menuBar;
    IStaticText
        statusLine,
        hello;
    IInfoArea
        infoArea;
    ACommandHandler
        commandHandler;
    // ...
};

```

The sample creates the menu bar as a child of the main window in the `AHelloWindow` constructor. The following excerpt is from `ahellow3.cpp`:

```

AHelloWindow :: AHelloWindow(const unsigned long windowId)
    : IFrameWindow(IFrameWindow::defaultStyle() |
                  IFrameWindow::minimizedIcon,
                  windowId)
    , menuBar(windowId, this)
    , statusLine(WND_STATUS, this, this)
    , hello(WND_HELLO, this, this)
    , infoArea(this)
    , commandHandler(this)
{
    // ...
}

```

3. Define a command handler to handle events created when a user selects a menu item. This sample derives a command handler class from `ICommandHandler` called `ACommandHandler`. The constructor for `ACommandHandler` uses a reference to `AHelloWindow` to access member functions of `AHelloWindow`, namely the function that aligns the text string in the client window. The following excerpt is from `ahellow3.hpp`:


```

class ACommandHandler : public ICommandHandler
{
public:
    ACommandHandler(AHelloWindow *helloFrame);
    virtual ~ACommandHandler() { };
protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    AHelloWindow *frame;
};

```

4. Override the `ICommandHandler::command` virtual function. When the user selects one of the commands from the menu bar the `ACommandHandler::command` function is called. This command calls `AHelloWindow::setTextAlignment` to align the text string depending on the ID of the event (`MI_CENTER`, `MI_LEFT`, or `MI_RIGHT`). The `ACommandHandler::command` function returns true to indicate that no other event handlers should process the menu bar command.

```

bool ACommandHandler :: command(ICommandEvent & cmdEvent)
{
    // Assume event will be processed
    bool eventProcessed(true);
    // Depending on the command event ID, call the
    // AHelloWindow::setAlignment function with the appropriate
    // AHelloWorld::Alignment value.
    switch (cmdEvent.commandId())
    {
    case MI_CENTER:
        frame->setTextAlignment(AHelloWindow::center);
        break;
    case MI_LEFT:
        frame->setTextAlignment(AHelloWindow::left);
        break;
    case MI_RIGHT:
        frame->setTextAlignment(AHelloWindow::right);
        break;
    default:
        // The event was not processed
        eventProcessed=false;
    } /* end switch */
    return(eventProcessed);
}

```

5. Enable your application to process events. The sample creates an `ACommandHandler` object called `commandHandler` as a child of `AHelloWindow`. The constructor then attaches the command handler to `AHelloWindow` by calling the `ICommandHandler::handleEventsFor` function. The following excerpt is from `ahellow3.cpp`:

```

AHelloWindow :: AHelloWindow(const unsigned long windowId)
    :IFrameWindow(IFrameWindow::defaultStyle() |
                  IFrameWindow::minimizedIcon,
                  windowId)
    ,menuBar(windowId, this)
    ,statusLine(WND_STATUS,this,this)
    ,hello(WND_HELLO, this, this)
    ,infoArea(this)
    ,commandHandler(this)
{
    // ...
    commandHandler.handleEventsFor(this);
    // ...
}

```

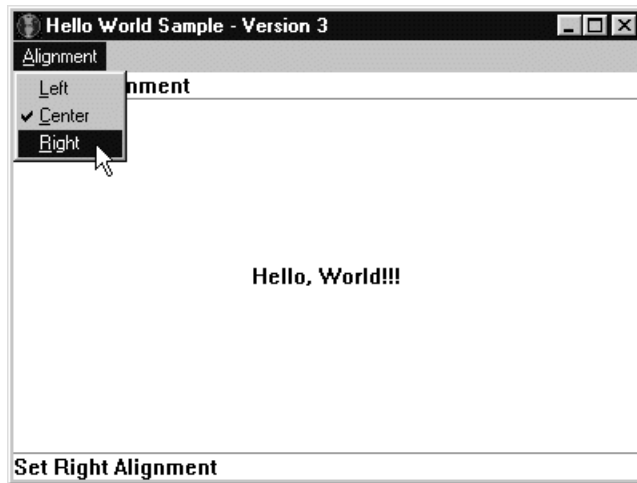
To stop event handling, the sample calls `ICommandHandler::stopHandlingEventsFor` in the destructor of the main window. The following is from `ahellow3.cpp`:

```

AHelloWindow :: ~AHelloWindow()
{
    commandHandler.stopHandlingEventsFor(this);
}

```

The following, from Windows, shows the menu generated by the Hello World version 3 sample:



RELATED CONCEPTS

“Chapter 4. Menus” on page 55

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Hello World Version 3: Add Command Handlers and Menu Bars” on page 412

“Add Menus to Your Application” on page 57

“Create a Frame Window” on page 4

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create a Pop-Up Menu

This section shows you two ways to create a pop-up menu. You can create a pop-up menu statically as a data member of a window class. You may also dynamically create one whenever the user requests a pop-up menu. Typically, you choose one of these approaches based on resource balancing. Static pop-up menus are only created and deleted once, but take up storage whether they are needed or not; dynamic pop-up menus are created and deleted on demand, which can slow processing if you request them frequently.

To create a pop-up menu, follow these steps:

1. Define the pop-up menu like a menu bar, either by using a resource script file or by using the `IMenu` member functions. See the topic “Add Menus to Your Application” on page 57 for more information.
2. Create a class derived from `IMenuHandler` and override the `IMenuHandler::makePopUpMenu()` member function. The `makePopUpMenu()` member function is called whenever the user requests a pop-up menu, usually by clicking mouse button 2, in a window for which menu requests are being

handled. In your implementation of this function, create and show the appropriate pop-up menu.. To display the pop-up menu, call `IMenu::show()`.

Version 6 of the Hello World application creates two pop-up menus: one for the "Hello, World" static text control and one for the Earth window drawing canvas. The application creates the the pop-up menu for the "Hello, World" static text control statically, and creates the other pop-up menu dynamically.

The contents of the pop-up menus are defined in the `ahellow6.rc` resource file as follows:

```
MENU WND_HELLOPOPUP
BEGIN
    MENUITEM "~Left-align text" , MI_LEFT
    MENUITEM "~Center text" , MI_CENTER
    MENUITEM "~Right-align text" , MI_RIGHT
END
MENU WND_EARTHPOPUP
BEGIN
    MENUITEM "~Twinkling stars" , MI_TWINKLE
    MENUITEM SEPARATOR
    MENUITEM "~Brighten stars" , MI_BRIGHT
    MENUITEM "~Dim stars" , MI_DIM
END
```

► OS/2

► AIX

This is an example of an OS/2 or AIX resource file.

In the `ahellow6.hpp` file, an `APopUpHandler` class is defined to process requests for making the pop-up menus appear, as follows:

```
class APopUpHandler : public IMenuHandler
{
public:
    APopUpHandler(IPopUpMenu* argPopUpMenu)
        : passedPopUpMenu(argPopUpMenu) {}
protected:
    virtual bool
        makePopUpMenu(IMenuEvent& menuEvent);
private:
    IPopUpMenu* passedPopUpMenu;
};
```

The `APopUpHandler` class derives from `IMenuHandler` and overrides the `IMenuHandler::makePopUpMenu()` function. You need the `APopUpHandler` constructor and the pointer to an `IPopUpMenu` object only if you want to create a pop-up menu statically. This constructor and pointer allows you to access your predefined `IPopUpMenu` object from within `makePopUpMenu()`. You do not need this constructor and pointer if you want to create dynamic pop-up menus since you will create your own `IPopUpMenu` object from each time within `makePopUpMenu()`.

The `makePopUpMenu()` function in Hello World version 6 demonstrates how to create a pop-up menu as a data member of the `AHelloWindow` class and how to dynamically show an existing pop-up menu using a data member.

The following sample is from the `ahellow6.cpp` file:

```

bool APopUpHandler::makePopUpMenu(IMenuEvent &menuEvent)
{
    bool eventProcessed(true);
    IPopUpMenu *popUpMenu;
    IWindow      *popUpOwner = menuEvent.controlWindow();
    IStaticText  *hello;
    AEarthWindow *earth;
    switch (popUpOwner->id())
    {
        case WND_HELLO:
            popUpMenu = passedPopUpMenu;
            if (popUpMenu)
            {
                hello = (IStaticText *)popUpOwner;
                popUpMenu->enableItem(MI_LEFT,
                                     hello->alignment() != IStaticText::centerLeft);
                popUpMenu->enableItem(MI_CENTER,
                                     hello->alignment() != IStaticText::centerCenter);
                popUpMenu->enableItem(MI_RIGHT,
                                     hello->alignment() != IStaticText::centerRight);
            }
            else eventProcessed = false;
            break;
        case WND_EARTH:
            popUpMenu = new IPopUpMenu(WND_EARTHPOPUP, popUpOwner);
            if (popUpMenu)
            {
                popUpMenu->setAutoDeleteObject();
                earth = (AEarthWindow *)popUpOwner;
                popUpMenu->checkItem(MI_TWINKLE, earth->isTwinkling());
                if (earth->isBright())
                { popUpMenu->disableItem(MI_BRIGHT); }
                else
                { popUpMenu->disableItem(MI_DIM); }
            }
            else eventProcessed=false;
            break;
        default:
            eventProcessed=false;
    }
    if (eventProcessed)
        popUpMenu->show(menuEvent.mousePosition());
    return(eventProcessed);
}

```

The case statement for WND_HELLO shows an existing pop-up menu that was created statically in the APopUpHandler constructor. a pop-up window statically. The code simply accesses the pop-up menu (called passedPopUpMenu in this sample) from within makePopUpMenu().

The case statement for WND_EARTH case creates a pop-up menu dynamically. The IPopUpMenu's constructor needs two arguments: the window ID (called WND_EARTHPOPUP in this sample) and a pointer to the owner of the pop-up menu. The IMenuEvent object stores a pointer to a window that originated the event. The application uses this pointer to construct the IPopUpMenu object. You access this pointer with the controlWindow() function. Because this application creates the menu using the new operator, it must also be deleted. The easiest way to delete a dynamic pop-up menu is to use the IPopUpMenu::setAutoDeleteObject function which causes the menu to be automatically deleted when the menu ends.

In either case, when the menu is found or created, the `makePopupMenu()` function displays the menu by using `IPopUpMenu::show()`. The mouse pointer position, which is taken from the menu event, is passed as one argument to specify where the pop-up menu will appear.

The `makePopupMenu()` function is only called for windows that are attached to the pop-up menu handler. Hello World version 6 attaches a pop-up menu handler directly to the `IStaticText` object `hello` and the `IDrawingCanvas` `earthWindow`. It attaches to the objects instead of the frame because these objects do not pass events up the owner chain. Therefore, use this approach if you develop portable applications. For the same reason, the command handler attaches to these objects so that the command events that result from using the pop-up menus will be sent to the command handler. In Hello World version 6, this is done using the following code from the `ahellow6.cpp` file:

```
commandHandler.handleEventsFor(&hello);
commandHandler.handleEventsFor(&earthWindow);
popUpHandler.handleEventsFor(&hello);
popUpHandler.handleEventsFor(&earthWindow);
```

By reusing the existing command handler, commands such as `MI_LEFT` can have the same processing whether they are generated by a menu bar item, a push button, an accelerator key, or a pop-up menu.

RELATED CONCEPTS

- “Chapter 4. Menus” on page 55
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Create a Frame Window” on page 4
- “Add Menus to Your Application” on page 57
- “Add a Menu Bar” on page 59
- “Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

Create a Pop-Up Menu for a List Box

The following example shows you how to create a pop-up menu for a list box:

1. Define your pop up menu in a resource script file similar to the one that follows. This excerpt is from `popmenu.rc`:

```
ID_POPUP2 MENUEX
BEGIN
    POPUP "&Color",                ID_COLOR
    BEGIN
        MENUITEM "Blue",          ID_BLUE
        MENUITEM "Green",         ID_GREEN
        MENUITEM "Red",           ID_RED
    END
    POPUP "&Text",                ID_TEXTMENU
    BEGIN
        MENUITEM "&Add Learn Text", ID_ADD_FIRST
        MENUITEM "&Delete All Text", ID_DELETE_TEXT
        MENUITEM "&Add Know Text",  ID_ADD_LAST
    END
END
```

This is from a Windows resource script file.

2. Declare a handler that is derived from both `IMenuHandler` and `ICommandHandler` so that the same handler can be used for handling the pop-up menu and the commands that originate from it. The following excerpt is from `popmenu.hpp`:

```
// Define the frame window
class AppWindow : public IFrameWindow
{
public:
    AppWindow(unsigned long windowId);
    virtual AppWindow();
private:
    ITitle        title;
    IMenuBar      menu;
    IMultiCellCanvas canvas;
    IStaticText   sttxt;
    IListBox      listbox;
    PopUpHandler  pLBPopUp;
};
```

Add an `IPopUpMenu` data member to the handler class as follows:

```
// Define the pop-up and command handler
class PopUpHandler : public IMenuHandler,
                    public ICommandHandler
{
public:
    PopUpHandler(IListBox & lbUpdate, IStaticText & stMsg);
    ~PopUpHandler();
protected:
    virtual bool makePopUpMenu(IMenuEvent& menuEvent);
    virtual bool command(ICommandEvent& cmdevt);
private:
    void setLBColor(unsigned long ulNewColor);
    void setLBText(unsigned long ulNewSize);
    IListBox * pLB;
    IStaticText * pST;
    IPopUpMenu * pPopUpMenu;
    unsigned long ulColor;
    unsigned long ulText;
};
```

3. Create an instance of the pop-up handler as shown in the following code. This excerpt is from `popmenu.cpp`:

```
// Construct the frame window with children
AppWindow::AppWindow(unsigned long windowId)
    : IFrameWindow(windowId, defaultStyle()),
      title(this, "PopUp", "Example"),
      canvas(ID_CANVAS, this, this),
      listbox(ID_LISTBX, &canvas, &canvas),
      sttxt(ID_TEXT, &canvas, &canvas),
      pLBPopUp(listbox, sttxt)
{
    // ...
}
```

4. Construct the pop-up handler. The following is the `PopUpHandler` constructor. It attaches a command handler and a menu handler to the pop-up menu by calling the appropriate `handleEventsFor` functions.

To handle command events that result from selecting items from the pop-up menu, override the `ICommandHandler::command` virtual function.

To display a pop-up menu override the `IMenuHandler::makePopupMenu` virtual function. Within `makePopupMenu`, you can either create and show a pop-up menu, or show an existing pop-up menu. This example shows an existing pop-up menu by calling `IPopUpMenu::show`. To get the current position of the mouse call `IMenuEvent::mousePosition`. Use this value as an argument to `IPopUpMenu::show` to display the pop-up menu at the current mouse position.

The following excerpt is from `popmenu.cpp`:

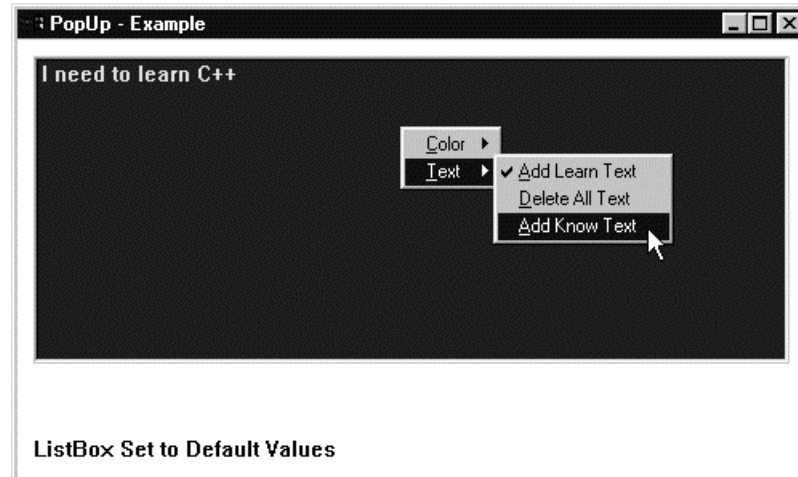
```
// PopUpHandler constructor
PopUpHandler::PopUpHandler(IListBox & LBUpdate, IStaticText & stMsg)
: pLB(&LBUpdate),
  pST(&stMsg),
  pPopupMenu(ID_POPUP2, pLB),
  ulColor(0),
  ulText(0)
{
    unsigned long ulColor, ulText;
    // We need pointers to list box and static text so we can
    // update them.
    // pLB points to the list box
    // pST points to the static text
    // Default color and alignment values
    ulColor = ID_BLUE;
    ulText = ID_ADD_FIRST;
    // Check the default items
    pPopupMenu.checkItem(ulColor);
    pPopupMenu.checkItem(ulText);
    // Set default color and alignment
    setLBColor(ulColor);
    setLBText(ulText);
    // handle events on pop-up menu
    ICommandHandler::handleEventsFor(pLB);
    IMenuHandler::handleEventsFor(pLB);
}
// ...
// Show the pop-up menu created in the constructor
bool PopUpHandler::makePopupMenu(IMenuEvent& menuEvt)
{
    pPopupMenu->show(menuEvt.mousePosition());
    return true;
}
// Command handling for the pop-up menu
bool PopUpHandler::command(ICommandEvent& cmdevt)
{
    switch (cmdevt.commandId())
    {
        case ID_RED:
        case ID_GREEN:
        case ID_BLUE:
            setLBColor(cmdevt.commandId());
            return true;
        case ID_ADD_FIRST:
        case ID_DELETE_TEXT:
        case ID_ADD_LAST:
            setLBText(cmdevt.commandId());
            return true;
    }
    return false;
}
// ...
// Handle the menu functions to change the list box color
void PopUpHandler::setLBColor(unsigned long ulNewColor)
{
    switch (ulNewColor) {
        case ID_RED:
```

```

        pLB->setBackgroundColor(IColor::kRed);
        pST->setText("Changed color to Red");
        pPopupMenu->checkItem(ID_RED);
        pPopupMenu->uncheckItem(ID_GREEN);
        pPopupMenu->uncheckItem(ID_BLUE);
        break;
    case ID_GREEN:
        pLB->setBackgroundColor(IColor::kGreen);
        pST->setText("Changed color to Green");
        pPopupMenu->uncheckItem(ID_RED);
        pPopupMenu->checkItem(ID_GREEN);
        pPopupMenu->uncheckItem(ID_BLUE);
        break;
    case ID_BLUE:
        pLB->setBackgroundColor(IColor::kBlue);
        pST->setText("Changed color to Blue");
        pPopupMenu->uncheckItem(ID_RED);
        pPopupMenu->uncheckItem(ID_GREEN);
        pPopupMenu->checkItem(ID_BLUE);
        break;
    default:
        break;
}
}
// Handle the menu functions to add text to the list box
void PopUpHandler::setLBText(unsigned long ulNewText)
{
    switch (ulNewText) {
    case ID_ADD_FIRST:
        pLB->addAscending("I need to learn C++");
        pST->setText("Added text to list box in ascending order");
        pPopupMenu.checkItem(ID_ADD_FIRST);
        pPopupMenu.uncheckItem(ID_DELETE_TEXT);
        pPopupMenu.uncheckItem(ID_ADD_LAST);
        break;
    case ID_DELETE_TEXT:
        pLB->removeAll();
        pST->setText("Deleted All Text from ListBox");
        pPopupMenu.uncheckItem(ID_ADD_FIRST);
        pPopupMenu.checkItem(ID_DELETE_TEXT);
        pPopupMenu.uncheckItem(ID_ADD_LAST);
        break;
    case ID_ADD_LAST:
        pLB->addDescending("I know C++");
        pST->setText("Added text to list box in descending order");
        pPopupMenu.uncheckItem(ID_ADD_FIRST);
        pPopupMenu.uncheckItem(ID_DELETE_TEXT);
        pPopupMenu.checkItem(ID_ADD_LAST);
        break;
    default:
        break;
    }
}
}

```

The following, from Windows, shows the pop-up menu generated from this example:



You can find the complete source to this example in the following files:

- popmenu.cpp
- popmenu.h
- popmenu.hpp
- popmenu.rc

RELATED CONCEPTS

“Chapter 4. Menus” on page 55 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Pop-Up Menu” on page 62
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445
 “File: popmenu.cpp”
 “File: popmenu.h” on page 72 “File: popmenu.hpp” on page 73
 “File: popmenu.rc” on page 74

Pop-Up Menu Example

File: popmenu.cpp

```
#include "popmenu.h"
#include "popmenu.hpp"
/*****
* main
* - Application entry point.
*****/
int main()
{
    AppWindow mainWindow(ID_MAIN);
    mainWindow.setSize(ISize(500,300));
    mainWindow.setFocus();
    mainWindow.show();
    IApplication::current().run();
    return 0;
}
/*****
* Class AViewWindow :: AViewWindow
* - Constructor for the main window.
*****/
```

```

*****/
AppWindow::AppWindow(unsigned long windowId)
: IFrameWindow(windowId, defaultStyle()),
  title(this, "PopUp","Example"),
  canvas(ID_CANVAS, this, this),
  listBox(ID_LISTBX, &canvas, &canvas),
  sttxt(ID_TEXT, &canvas, &canvas),
  pLBPopUp(listBox, sttxt)
{
  listBox.setForegroundColor(IColor::kYellow);
  sttxt.setText("ListBox Set to Default Values");
  sttxt.setLimit(30);
  // Customize the multicell canvas
  setClient(&canvas);
  canvas.addToCell(&listBox, 2, 2);
  canvas.setRowHeight(2, 20, true);
  canvas.setColumnWidth(2, 20, true);
  canvas.addToCell(&sttxt, 2, 6);
  canvas.setRowHeight(1, 10);
  canvas.setRowHeight(4, 25);
  canvas.setRowHeight(7, 10);
  canvas.setColumnWidth(1, 10);
  canvas.setColumnWidth(3, 10);
}
/*****
* Class PopUpHandler :: PopUpHandler *
*****/
PopUpHandler::PopUpHandler(IListBox & LBUpdate, IStaticText & stMsg)
: pLB(&LBUpdate),
  pST(&stMsg),
  pPopUpMenu(ID_POPUP2, pLB),
  ulColor(0),
  ulText(0)
{
  unsigned long ulColor, ulText;
  // We need pointers to list box and static text so we can
  // update them.
  // pLB points to the list box
  // pST points to the static text
  // Default color and alignment values
  ulColor = ID_BLUE;
  ulText = ID_ADD_FIRST;
  // Check the default items
  pPopUpMenu.checkItem(ulColor);
  pPopUpMenu.checkItem(ulText);
  // Set default color and alignment
  setLBColor(ulColor);
  setLBText(ulText);
  // handle events on pop-up menu
  ICommandHandler::handleEventsFor(pLB);
  IMenuHandler::handleEventsFor(pLB);
}
/*****
* Class PopUpHandler :: ~PopUpHandler *
*****/
PopUpHandler::~PopUpHandler()
{
  ICommandHandler::stopHandlingEventsFor(pLB);
  IMenuHandler::stopHandlingEventsFor(pLB);
  pLB = 0;
  pST = 0;
}
/*****
* Class PopUpHandler :: setLBColor *
*****/
void PopUpHandler::setLBColor(unsigned long ulNewColor)
{

```

```

switch (ulNewColor) {
case ID_RED:
    pLB->setBackgroundColor(IColor::kRed);
    pST->setText("Changed color to Red");
    pPopupMenu.checkItem(ID_RED);
    pPopupMenu.uncheckItem(ID_GREEN);
    pPopupMenu.uncheckItem(ID_BLUE);
    break;
case ID_GREEN:
    pLB->setBackgroundColor(IColor::kGreen);
    pST->setText("Changed color to Green");
    pPopupMenu.uncheckItem(ID_RED);
    pPopupMenu.checkItem(ID_GREEN);
    pPopupMenu.uncheckItem(ID_BLUE);
    break;
case ID_BLUE:
    pLB->setBackgroundColor(IColor::kBlue);
    pST->setText("Changed color to Blue");
    pPopupMenu.uncheckItem(ID_RED);
    pPopupMenu.uncheckItem(ID_GREEN);
    pPopupMenu.checkItem(ID_BLUE);
    break;
default:
    break;
}
}
/*****
* Class PopUpHandler :: setLBText
*****/
void PopUpHandler::setLBText(unsigned long ulNewText)
{
    switch (ulNewText) {
    case ID_ADD_FIRST:
        pLB->addAscending("I need to learn C++");
        pST->setText("Added text to list box in ascending order");
        pPopupMenu.checkItem(ID_ADD_FIRST);
        pPopupMenu.uncheckItem(ID_DELETE_TEXT);
        pPopupMenu.uncheckItem(ID_ADD_LAST);
        break;
    case ID_DELETE_TEXT:
        pLB->removeAll();
        pST->setText("Deleted All Text from ListBox");
        pPopupMenu.uncheckItem(ID_ADD_FIRST);
        pPopupMenu.checkItem(ID_DELETE_TEXT);
        pPopupMenu.uncheckItem(ID_ADD_LAST);
        break;
    case ID_ADD_LAST:
        pLB->addDescending("I know C++");
        pST->setText("Added text to list box in descending order");
        pPopupMenu.uncheckItem(ID_ADD_FIRST);
        pPopupMenu.uncheckItem(ID_DELETE_TEXT);
        pPopupMenu.checkItem(ID_ADD_LAST);
        break;
    default:
        break;
    }
}
/*****
* Class PopUpHandler :: makePopupMenu
*****/
bool PopUpHandler::makePopupMenu(IMenuEvent& menuEvt)
{
    pPopupMenu.show(menuEvt.mousePosition());
    return true;
}
/*****
* Class PopUpHandler :: command
*****/

```

```

*****/
bool PopUpHandler::command(ICommandEvent& cmdevt)
{
    switch (cmdevt.commandId())
    {
        case ID_RED:
        case ID_GREEN:
        case ID_BLUE:
            setLBColor(cmdevt.commandId());
            return true;
        case ID_ADD_FIRST:
        case ID_DELETE_TEXT:
        case ID_ADD_LAST:
            setLBText(cmdevt.commandId());
            return true;
    }
    return false;
}

```

RELATED CONCEPTS

“Chapter 4. Menus” on page 55
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Pop-Up Menu for a List Box” on page 65
 “File: popmenu.h” “File: popmenu.hpp” on page 73
 “File: popmenu.rc” on page 74
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

File: popmenu.h

```

#ifndef _POPMENU_H_
#define _POPMENU_H_
/*****
 * Window IDs
 *****/
#define ID_MAIN 10
#define ID_CANVAS 20
#define ID_TEXT 30
#define ID_LISTBX 40
/*****
 * Pop-Up Menu IDs
 *****/
#define ID_POPUP2 300
#define ID_COLOR 500
#define ID_BLUE 510
#define ID_GREEN 520
#define ID_RED 530
#define ID_TEXTMENU 600
#define ID_ADD_FIRST 610
#define ID_DELETE_TEXT 620
#define ID_ADD_LAST 630
#endif

```

RELATED CONCEPTS

“Chapter 4. Menus” on page 55
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Pop-Up Menu for a List Box” on page 65

“File: popmenu.cpp” on page 69

“File: popmenu.h” on page 72“File: popmenu.hpp”

“File: popmenu.rc” on page 74

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: popmenu.hpp

```
#ifndef _POPMENU_
#define _POPMENU_
#include <iframe.hpp>
#include <icmdhdr.hpp>
#include <iapp.hpp>
#include <ititle.hpp>
#include <istattxt.hpp>
#include <icanvas.hpp>
#include <imenubar.hpp>
#include <imcelcv.hpp>
#include <ientryfd.hpp>
#include <imenuhdr.hpp>
#include <ilistbox.hpp>
#include <icolor.hpp>
#include <ipopmenu.hpp>
class AppWindow;
class PopUpHandler;
/*****
 * Class AppWindow
 *****/
class AppWindow : public IFrameWindow
{
public:
    AppWindow(unsigned long windowId);
private:
    ITitle        title;
    IMultiCellCanvas canvas;
    IStaticText   sttxt;
    IListBox      listBox;
    PopUpHandler  pLBPopUp;
};
/*****
 * Class PopUpHandler
 *****/
class PopUpHandler : public IMenuHandler,
                    public ICommandHandler
{
public:
    PopUpHandler(IListBox & lbUpdate, IStaticText & stMsg);
    virtual ~PopUpHandler();
protected:
    virtual bool makePopUpMenu(IMenuEvent& menuEvent);
    virtual bool command(ICommandEvent& cmdevt);
private:
    void setLBColor(unsigned long ulNewColor);
    void setLBText(unsigned long ulNewSize);
    IListBox * pLB;
    IStaticText * pST;
    IPopupMenu pPopupMenu;
    unsigned long ulColor;
    unsigned long ulText;
};
#endif
```

RELATED CONCEPTS

- “Chapter 4. Menus” on page 55
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Create a Pop-Up Menu for a List Box” on page 65
- “File: popmenu.cpp” on page 69
- “File: popmenu.h” on page 72“File: popmenu.rc”
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

File: popmenu.rc**WIN**

This is a Windows resource script file.

```
#include "popmenu.h"
ID_POPUP2 MENUEX
BEGIN
  POPUP "&Color",          ID_COLOR
  BEGIN
    MENUITEM "&Blue",      ID_BLUE
    MENUITEM "&Green",     ID_GREEN
    MENUITEM "&Red",       ID_RED
  END
  POPUP "&Text",          ID_TEXTMENU
  BEGIN
    MENUITEM "&Add Learn Text", ID_ADD_FIRST
    MENUITEM "&Delete All Text", ID_DELETE_TEXT
    MENUITEM "Add &Know Text",  ID_ADD_LAST
  END
END
```

AIX**OS/2**

This is an AIX or OS/2 resource script file:

```
#include "popmenu.h"
MENU ID_POPUP2
BEGIN
  SUBMENU "~Color",      ID_COLOR
  BEGIN
    MENUITEM "~Blue",    ID_BLUE
    MENUITEM "~Green",   ID_GREEN
    MENUITEM "~Red",     ID_RED
  END
  SUBMENU "~Text",      ID_TEXTMENU
  BEGIN
    MENUITEM "~Add Learn Text", ID_ADD_FIRST
    MENUITEM "~Delete All Text", ID_DELETE_TEXT
    MENUITEM "Add ~Know Text",  ID_ADD_LAST
  END
END
```

RELATED CONCEPTS

- “Chapter 4. Menus” on page 55
- “Chapter 21. Resources” on page 297
- “Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Pop-Up Menu for a List Box” on page 65

“Convert Application Resources” on page 395

“File: popmenu.cpp” on page 69

“File: popmenu.h” on page 72 “File: popmenu.hpp” on page 73

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Dynamically Modify a Menu with the ISubmenu Class

The ISubmenu class lets you dynamically modify the contents of a pull-down menu, cascade menu, or pop-up menu. Unlike the IMenu class, when the user closes the menu, the ISubmenu class will restore the changes you have made to the menu. You do not directly create ISubmenu objects. When a user displays a pull-down, cascade, or pop-up menu, the IMenuHandler class creates an ISubmenu object and passes it to the menuShowing() member function.

To use the ISubmenu class, create a new class derived from the IMenuHandler class. Override the IMenuHandler::menuShowing() member function to make changes to the ISubmenu object that was passed as a parameter. IMenuHandler calls this function before the pull-down, cascade, or pop-up menu appears on the screen. You can identify the menu being shown and represented by the ISubmenu object by calling IMenu::id() on the ISubmenu object or IMenuEvent::menuItemid().

The Multiline Edit Sample shows you how to use the ISubmenu class. The sample uses this class to disable or enable the **Cut**, **Copy**, and **Paste** menu choices depending on whether the user has selected any text in the multiline edit control, or whether the clipboard contains any text:

1. Derive your main window from the IMenuHandler class. The following code sample, from the amle.hpp file, declares the main window class AEditorWindow.

```
class AEditorWindow : public IFrameWindow
                    , public ICommandHandler
                    , public IMenuHandler
                    , public AUserMessageHandler
{
    // ...
}
```

This allows us to declare the IMenuHandler::menuShowing() member function as a protected member function in the AEditorWindow class as follows:

```
protected:
    bool command(ICommandEvent& cmdEvent);
    bool menuShowing(IMenuEvent& mnEvt
                    , ISubmenu& smnAboutToShow);
    bool userMessage(IEvent& evt);
    bool setFileMenuItemsState(bool f);
    AEditorWindow &displayLoadFailedMsg();
    bool loadMLE();
```

2. Enable your application to process events by attaching event handlers to your main window. The constructor for the AEditorWindow class calls the handleEventsFor so that the AEditorWindow can start processing menu events.

```
((ICommandHandler *)this)->handleEventsFor(this);
((AUserMessageHandler *)this)->handleEventsFor(this);
((IMenuHandler *)this)->handleEventsFor(this);
```

The AEditorWindow constructor calls the handleEventsFor function three times:

- a. Handle command events
- b. Handle user message events
- c. Handle menu events

The first call to `handleEventsFor` will process events with the `ICommandHandler::command` virtual function. The third call to `handleEventsFor` will process events with the `IMenuHandler::menuShowing` virtual functions. This sample uses the `ICommandHandler` class to process all menu events except for the **Edit** command. Instead, the `IMenuHandler` processes this command.

3. Override the `IMenuHandler::menuShowing` virtual function. This sample overrides the `menuShowing` virtual function to handle events created from the **Edit** command:

```
bool AEditorWindow::menuShowing( IMenuEvent& mnEvt,
                                ISubmenu& smnAboutToShow)
{
    bool fProcessed = false;
    switch (smnAboutToShow.id())
    {
        case MI_EDIT:
            if (!mle.hasSelectedText())
            {
                smnAboutToShow.disableItem(MI_COPY);
                smnAboutToShow.disableItem(MI_CUT);
                fProcessed = true;
            }
            try
            {
                if (!mle.clipboardHasTextFormat())
                {
                    smnAboutToShow.disableItem(MI_PASTE);
                    fProcessed = true;
                }
            }
            catch(IAccessError& exception)
            {
                smnAboutToShow.disableItem(MI_PASTE);
                fProcessed = true;
            }
            break;
        default:
            break;
    }
    return fProcessed;
}
```

`IMenuHandler` calls the `menuShowing` function after the user selects the **Edit** command, and before the application displays the pull-down menu.

The **Edit** pull-down menu has three commands to select from: **Cut**, **Copy**, and **Paste**. These commands are enabled before the application displays the pull-down menu. This `menuShowing` function disables these three commands depending on whether the multiline edit has text selected or the clipboard holds text. The overridden `menuShowing` function disables the commands with the `ISubmenu::disableItem` function (as opposed to the equivalent function in the `IMenuBar` class). Using the `ISubmenu` function in this case means that you do not have to call any functions to restore disabled command choices. The `ISubmenu` class will undo calls to the `disableItem` function when the **Edit** pull-down menu is dismissed, putting the menu back in its original state with all the menu items enabled.

You can find the Multiline Edit Sample in the `samples/ioc/mle` directory.

RELATED CONCEPTS

“Chapter 4. Menus” on page 55 “Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

Multiline Edit Sample

“Add Menus to Your Application” on page 57

“Add a Menu Bar” on page 59

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Chapter 5. Use Cursor Classes

Window classes that can contain one or more items generally provide a nested cursor class. The *cursor classes* provide member functions to move through the items, either forward or backward.

The Open Class Library cursor classes are designed to have the same look and feel as the collection class cursors.

A cursor must be in a valid state to access the items in a list. A cursor is generally created in an invalid state. Any cursor function that causes the cursor to point to an item in the list validates the cursor. For example, the function `setToFirst` causes the cursor to be valid if there are items in the list. If the contents of the list that the cursor is iterating through change by the addition or removal of items, the cursor becomes invalid. It cannot be used to access items in the list until it is validated again by a function that points the cursor at a valid item.

`IWindow::ChildCursor` is an exception to this.

Some cursors support iteration over items in a collection that match a particular property. For example, the constructor for a list box cursor can have a second parameter that determines whether the cursor returns all items in the list box or just the selected items.

The following example, from the Hello World version 5 `ahellow5.cpp` file, shows how to set text from the first selected item in a multiple-selection list box:

```
// ...
AHelloWindow &
AHelloWindow :: setTextFromListBox()
{
    // Create a cursor to the list box. Using the default
    // filter for a list box cursor, selectedItems, causes the
    // setToFirst() function to position the cursor to the
    // first selected item.
    // Set the hello IStaticText control text value.
    IListBox::Cursor lbCursor(listBox);
    lbCursor.setToFirst();
    hello.setText(listBox.elementAt(lbCursor));
    // Return a reference to the frame
    return (*this);
}
// ...
```

RELATED CONCEPTS

Iteration

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

IListBox

Chapter 6. Static Text Controls

Static text controls are text fields, bitmaps, icons, and boxes that you can use to label or box other controls. Your user typically does not interact with these controls using the keyboard or mouse. Generally, you do not need to change a static control's appearance on the screen, so visually these controls tend to be unchanging.

The static control classes include the following classes:

- `IStaticText` displays a text string.
- `IBitmapControl` displays a bitmap with optional text.
- `IIconControl` displays an icon with optional text.
- `IOutlineBox` displays a simple border.
- `IGroupBox` displays a border with a text label.

RELATED CONCEPTS

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a Static Text Control"

"Write an Event Handler" on page 25

"Task and Samples Cross-Reference Table" on page 445

Create a Static Text Control

Static text controls are text fields, bitmaps, icons, and boxes that you can use to label or box other controls. A user typically does not interact with these controls using the keyboard or mouse. Generally, you do not need to change a static control's appearance on the screen, so visually these controls tend to be unchanging. Static text control classes include `IBitmapControl`, `IIconControl`, `IOutlineBox`, `IGroupBox`, and `IStaticText`. The `IStaticText` class creates and manages the static text control window.

You can set the text and its color, size, and position in the static text window.

The following sample comes from the `ahellow1.cpp` file from the Hello World sample application and shows how to create a static text control:

```
IStaticText hello(WND_HELLO, &mainWindow, &mainWindow);  
hello.setText(STR_HELLO);  
hello.setAlignment(IStaticText::centerCenter);  
mainWindow.setClient(&hello);
```

The first line uses the window ID, the parent window, and the owner window to create the static text control and an object for it.

The second line sets a text string in the control using the `setText` member function, which is inherited from `ITextControl`. The string is provided in the resource (`.rc`) file.

The third line uses the `setAlignment` member function to center the static text.

The following figure, from Windows, shows the Hello World version 1 static text control:



RELATED CONCEPTS

- "Chapter 6. Static Text Controls" on page 81
- "Chapter 1. Windows" on page 1
- "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

- "Create a Static Text Control" on page 81
- "Create an Entry Field" on page 83
- "Create an MLE" on page 92
- "Hello World Version 1: Create a Main Window" on page 402
- "Write an Event Handler" on page 25
- "Task and Samples Cross-Reference Table" on page 445

Chapter 7. Entry Fields and Multiline Edit (MLE) Fields

An *entry field* is a control window that enables a user to view and edit a single line of text. An entry field provides the text-editing capabilities of a simple text editor and is useful whenever an application requires a short line of text from the user.

Use the `IEntryField` class to represent an entry field control.

Multiline Edit Fields

If the application requires more sophisticated text-editing capabilities and multiple lines of text from the user, the application can use a multiline edit (MLE) field.

A multiline edit field enables users to view and edit multiple lines of text. Use the `IMultiLineEdit` class to create an MLE field. The member functions of the `IMultiLineEdit` class enable you to display text files with horizontal and vertical scrolling, read a file into and save it from an MLE, or perform basic clipboard tasks (such as cut, copy, paste, and clear).

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create an Entry Field”

“Create an MLE” on page 92

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create an Entry Field

The following is an example of creating an entry field.

1. Declare three entry fields in the `.hpp` file. This example declares three entry fields called `ef1`, `ef2`, and `ef3`:

```
#include <ientryfd.hpp>
/*****
/* Create the frame window */
/* Declare the entry fields */
*****/
class AppWindow : public IFrameWindow
{
public:
    AppWindow(unsigned long windowId);
    AppWindow();
    void handleEvents(unsigned long eventtype);
private:
    ICanvas        canvas;
    IEntryField    ef1;
    IEntryField    ef2;
    IEntryField    ef3;
    ACommandHandler * commandHandler;
};
```

2. Construct the entry fields in the `.cpp` file and manipulate data inside them. The constructor for `ef3` takes `efStyle` as one of its arguments. This variable defines the following style attributes:

Style	Description
IWindow::visible	Causes the window to be visible.
IWindow::tabStop	Identifies the window as one the user can move the input focus to using the Tab key. <div style="background-color: #cccccc; padding: 2px; display: inline-block;">AIX</div> This style does has no affect on tabbing between windows. The Motif operating environment determines how the input focus is changed via the keyboard.
IWindow::group	Identifies the window as being the first in a group. Windows in a group are sibling windows consecutively ordered by sibling order. You can navigate between windows in a group using the keyboard arrow keys.
IEntryField::margin	Draws a border around the entry field, with a margin between the border and the entry field. The margin's size is determined by the current font being used for entry field text (half a character-width wide and half a character-height high). Because the margin and border are drawn around the entry field, using the margin style does not change the position of the entry field itself.
IEntryField::autoScroll	If the user tries to move off the end of a line, the entry field automatically scrolls one-third the width of the window in the appropriate direction.

The example uses the following member functions from IEntryField:

Member Function	Description
setLimit	Sets the maximum number of bytes the entry field can contain.
setText	Sets the control window's text. (This is an inherited function from ITextControl.)
setBackgroundColor	Sets the background color to the indicated color. (This is an inherited function from IWindow.)
setForegroundColor	Sets the foreground color to the indicated color. (This is an inherited function from IWindow.)
setBorderColor	Sets the border color to the indicated color. (This is an inherited function from IWindow.)
setFocus	Sets the input focus to the window. (This is an inherited function from IWindow.)

```
#include "entryf3.h"
#include "entryf3.hpp"
#include <icoordsy.hpp>
```



```

IEntryField::Style efStyle = IWindow::visible |
                             IWindow::tabStop |
                             IControl::group |
                             IEntryField::margin |
                             IEntryField::autoScroll;
/*****
* Class AppWindow :: AppWindow
* - Window constructor
*****/
AppWindow :: AppWindow(unsigned long windowId)
             : IFrameWindow("Entry Field Example",
                             windowId,
                             IFrameWindow::defaultStyle() |
                             IFrameWindow::menuBar),
             canvas(ID_CANVAS, this, this),
             ef1(ID_ENTRY, &canvas,&canvas,
                 IRectangle( 10, 200, 600, 240)),
             ef2(ID_ENTRY2, &canvas, &canvas,
                 IRectangle( 10, 125, 300, 175)),
             ef3(ID_ENTRY3, &canvas, &canvas,
                 IRectangle( 10, 50, 300, 100),
                 efStyle)
{
    // Create first entry field
    ef1.setLimit(50);
    ef1.setText("Initial Text for Entry Field 1");
    ef1.setBackgroundColor(IColor(213, 219, 255));
    ef1.setForegroundColor(IColor::kBlack);
    ef1.setBorderColor(IColor::kGreen);
    ef1.setFocus();
    // Create second entry field
    ef2.setLimit(50);
    ef2.setText("Initial Text for Entry Field 2");
    ef2.setBackgroundColor(IColor(213, 219, 255));
    ef2.setForegroundColor(IColor::kBlack);
    ef2.setBorderColor(IColor::kGreen);
    // Create third entry field
    ef3.setLimit(50);
    ef3.setText("Initial Text for Entry Field 3");
    ef3.setBackgroundColor(IColor(213, 219, 255));
    ef3.setForegroundColor(IColor::kBlack);
    ef3.setBorderColor(IColor::kGreen);
    moveSizeTo(IRectangle(0, 0, 670, 350));
    canvas.setBackgroundColor(IColor(170, 182, 255));
    setClient(&canvas);
    show();
    commandHandler = new ACommandHandler(this, &ef1,
                                         &ef2, &ef3);
    commandHandler->handleEventsFor(this);
}

```

3. Declare a command handler in the .hpp. This example will override the virtual function `ICommandHandler::command` to process events from the menu.

```

class ACommandHandler : public ICommandHandler {
public:
    ACommandHandler(AppWindow *efWindow,
                   IEntryField *ef1,
                   IEntryField *ef2,
                   IEntryField *ef3);

protected:
    virtual bool command(ICommandEvent& cmdEvent);

private:
    AppWindow *ef;
    IEntryField *ef1,
                *ef2,
                *ef3;
};

```

4. Add event handling in the .cpp to manipulate the data inside the entry fields. The menu of this example provides you with five commands:

- **Toggle Read Only for 1**
- **Cut Selected from 2**
- **Copy Selected from 1**
- **Paste into 3**
- **Clear Selected from 3**

The example overrides the `ICommandHandler::command` function to process these commands:

```
ACommandHandler::ACommandHandler(AppWindow *efWindow,
                                IEntryField *ef1,
                                IEntryField *ef2,
                                IEntryField *ef3):
    ef(efWindow),
    ef1(ef1),
    ef2(ef2),
    ef3(ef3)
{
}
bool ACommandHandler::command(ICommandEvent& cmdEvent)
{
    switch (cmdEvent.commandId())
    {
        case ID_READONLY_ITEM:
            ef1->enableDataUpdate(ef1->isWriteable());
            break;
        case ID_COPY_ITEM:
            if (ef1->hasSelectedText()) {
                ef1->copy();
            }
            break;
        case ID_CUT_ITEM:
            if (ef2->hasSelectedText()) {
                ef2->cut();
            }
            break;
        case ID_PASTE_ITEM:
            if (ef3->clipboardHasTextFormat()) {
                ef3->paste();
            }
            break;
        case ID_CLEAR_ITEM:
            if (ef3->hasSelectedText()) {
                ef3->clear();
            }
            break;
        return true;
    } // end of switch
    return false;
}
```

The following figure, from Windows, shows the entry fields created using the preceding example:



You can find the complete source to this example in the following files:

- entryf3.cpp
- entryf3.h
- entryf3.hpp
- entryf3.rc

RELATED CONCEPTS

“Chapter 7. Entry Fields and Multiline Edit (MLE) Fields” on page 83

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Static Text Control” on page 81

“Create an MLE” on page 92

“File: entryf3.cpp”

“File: entryf3.h” on page 89

“File: entryf3.hpp” on page 90

“File: entryf3.rc” on page 91

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Entry Field Example

File: entryf3.cpp

```
#include "entryf3.h"
#include "entryf3.hpp"
#include <icoordsy.hpp>
IEntryField::Style efStyle = IWindow::visible |
                             IWindow::tabStop |
                             IControl::group |
                             IEntryField::margin |
                             IEntryField::autoScroll;
/*****
* Class AppWindow :: AppWindow
* - Window constructor
*****/
AppWindow :: AppWindow(unsigned long windowId)
    : IFrameWindow("Entry Field Example",
```

```

        windowId,
        IFrameWindow::defaultStyle() |
        IFrameWindow::menuBar),
canvas(ID_CANVAS, this, this),
ef1(ID_ENTRY, &canvas,&canvas,
    IRectangle( 10, 200, 600, 240)),
ef2(ID_ENTRY2, &canvas, &canvas,
    IRectangle( 10, 125, 300, 175)),
ef3(ID_ENTRY3, &canvas, &canvas,
    IRectangle( 10, 50, 300, 100),
    efStyle)
{
    // Create first entry field
    ef1.setLimit(50);
    ef1.setText("Initial Text for Entry Field 1");
    ef1.setBackgroundColor(IColor(213, 219, 255));
    ef1.setForegroundColor(IColor::kBlack);
    ef1.setBorderColor(IColor::kGreen);
    ef1.setFocus();
    // Create second entry field
    ef2.setLimit(50);
    ef2.setText("Initial Text for Entry Field 2");
    ef2.setBackgroundColor(IColor(213, 219, 255));
    ef2.setForegroundColor(IColor::kBlack);
    ef2.setBorderColor(IColor::kGreen);
    // Create third entry field
    ef3.setLimit(50);
    ef3.setText("Initial Text for Entry Field 3");
    ef3.setBackgroundColor(IColor(213, 219, 255));
    ef3.setForegroundColor(IColor::kBlack);
    ef3.setBorderColor(IColor::kGreen);
    moveSizeTo(IRectangle(0, 0, 670, 350));
    canvas.setBackgroundColor(IColor(170, 182, 255));
    setClient(&canvas);
    show();
    commandHandler = new ACommandHandler(this, &ef1,
                                         &ef2, &ef3);
    commandHandler->handleEventsFor(this);
}
/*****
* Class AppWindow :: AppWindow
* - Window destructor
*****/
AppWindow :: AppWindow()
{
    commandHandler->stopHandlingEventsFor(this);
}
/*****
* Class ACommandHandler :: ACommandHandler
*****/
ACommandHandler::ACommandHandler(AppWindow *efWindow,
                                 IEntryField *ef1,
                                 IEntryField *ef2,
                                 IEntryField *ef3):
    ef(efWindow),
    ef1(ef1),
    ef2(ef2),
    ef3(ef3)
{
}
/*****
* Class ACommandHandler :: command
*****/
bool ACommandHandler::command(ICommandEvent& cmdEvent)
{
    switch (cmdEvent.commandId())
    {

```

```

    case ID_READONLY_ITEM:
        ef1->enableDataUpdate(!ef1->isWritable());
        break;
    case ID_COPY_ITEM:
        if (ef1->hasSelectedText()) {
            ef1->copy();
        }
        break;
    case ID_CUT_ITEM:
        if (ef2->hasSelectedText()) {
            ef2->cut();
        }
        break;
    case ID_PASTE_ITEM:
        if (ef3->clipboardHasTextFormat()) {
            ef3->paste();
        }
        break;
    case ID_CLEAR_ITEM:
        if (ef3->hasSelectedText()) {
            ef3->clear();
        }
        break;
        return true;
    }
    return false;
}
/*****
* main
* - Application entry point.
*****/
int main()
{
    ICoordinateSystem::setApplicationOrientation(
        ICoordinateSystem::kOriginLowerLeft);
    AppWindow appWindow(WID_MAIN);
    IApplication::current().run();
    return 0;
}

```

RELATED CONCEPTS

“Chapter 7. Entry Fields and Multiline Edit (MLE) Fields” on page 83

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create an Entry Field” on page 83

“File: entryf3.h”

“File: entryf3.hpp” on page 90

“File: entryf3.rc” on page 91

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: entryf3.h

```

#ifndef ENTRYFD_H
#define ENTRYFD_H
#define WID_MAIN 1000 // Frame Window ID
#define ID_CANVAS 1001 // Canvas ID
#define ID_ENTRY 1003 // Entry Field ID
#define ID_ENTRY2 1004 // Entry Field2 ID
#define ID_ENTRY3 1005 // Entry Field3 ID
#define MMI_ACTIONS 1999 // Manipulate menu item ID
#define ID_CENTER_ITEM 2000 // Center menu item ID

```

```

#define ID_LEFT_ITEM      2001
#define ID_RIGHT_ITEM     2002
#define ID_READONLY_ITEM 2003
#define ID_CUT_ITEM       2004
#define ID_COPY_ITEM      2005
#define ID_PASTE_ITEM     2006
#define ID_CLEAR_ITEM     2007
#endif

```

RELATED CONCEPTS

“Chapter 7. Entry Fields and Multiline Edit (MLE) Fields” on page 83

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create an Entry Field” on page 83

“File: entryf3.cpp” on page 87

“File: entryf3.hpp”

“File: entryf3.rc” on page 91

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: entryf3.hpp

```

#include <iframe.hpp>
#include <icmdhdr.hpp>
#include <icanvas.hpp>
#include <istattxt.hpp>
#include <ientryfd.hpp>
#include <iapp.hpp>
class AppWindow;
/*****
 * Class ACommandHandler
 *****/
class ACommandHandler : public ICommandHandler
{
public:
    ACommandHandler(AppWindow *efWindow,
                    IEntryField *ef1,
                    IEntryField *ef2,
                    IEntryField *ef3);

protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    AppWindow *ef;
    IEntryField *ef1,
                *ef2,
                *ef3;
};
/*****
 * Class AppWindow
 *****/
class AppWindow : public IFrameWindow
{
public:
    AppWindow(unsigned long windowId);
    AppWindow();
    void handleEvents(unsigned long eventtype);
private:
    ICanvas canvas;
    IEntryField ef1;
};

```

```

    IEntryField    ef2;
    IEntryField    ef3;
    ACommandHandler * commandHandler;
};

```

RELATED CONCEPTS

“Chapter 7. Entry Fields and Multiline Edit (MLE) Fields” on page 83

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create an Entry Field” on page 83

“File: entryf3.cpp” on page 87

“File: entryf3.h” on page 89

“File: entryf3.rc”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: entryf3.rc

WIN

This resource script file is for Windows:

```

#include "entryf3.h"
WID_MAIN MENUEX
BEGIN
    POPUP "&Manipulate"                , MMI_ACTIONS
    BEGIN
        MENUITEM "Toggle Read &Only for 1" , ID_READONLY_ITEM
        MENUITEM "Cu&t Selected from 2"    , ID_CUT_ITEM
        MENUITEM "&Copy Selected from 1"   , ID_COPY_ITEM
        MENUITEM "&Paste into 3"          , ID_PASTE_ITEM
        MENUITEM "&Clear Selected from 3"  , ID_CLEAR_ITEM
    END
END

```

AIX

OS/2

This resource script file is for AIX or OS/2:

```

#include "entryf3.h"
MENU WID_MAIN
BEGIN
    SUBMENU "~Manipulate"                , MMI_ACTIONS
    BEGIN
        MENUITEM "Toggle Read ~Only for 1" , ID_READONLY_ITEM
        MENUITEM "Cu t Selected from 2"    , ID_CUT_ITEM
        MENUITEM "~Copy Selected from 1"   , ID_COPY_ITEM
        MENUITEM "~Paste into 3"          , ID_PASTE_ITEM
        MENUITEM "~Clear Selected from 3"  , ID_CLEAR_ITEM
    END
END

```

RELATED CONCEPTS

“Chapter 7. Entry Fields and Multiline Edit (MLE) Fields” on page 83

“Chapter 21. Resources” on page 297

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create an Entry Field” on page 83

“Convert Application Resources” on page 395

“File: entryf3.cpp” on page 87

“File: entryf3.h” on page 89

“File: entryf3.hpp” on page 90

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create an MLE

A *multiline edit* (MLE) field enables users to view and edit multiple lines of text. Use the `IMultiLineEdit` class to create an MLE field. The member functions of the `IMultiLineEdit` class enable you to display text files with horizontal and vertical scrolling, read a file into and save it from an MLE, or perform basic clipboard tasks (for example, cut, paste, copy, and clear).

WIN

Some library styles and functions are ignored on Windows, for example `IMultiLineEdit::enableWordWrap`.

To create an object of the `IMultiLineEdit` class, include the ID of a specified MLE, the parent and owner windows, an `IRectangle` object, and one or more styles.

Styles define such functions as scrolling text, wrapping words, adding a border, and making the field read-only.

The following, from the Multiline Edit Sample, creates an MLE with basic file and clipboard operations. You can find this sample in the `samples/ioc/mle` directory.

1. Declare an application frame window that contains an MLE and a command handler that can process events for the MLE, as follows:

```
/* Create the command handler */
class ACommandHandler : public ICommandHandler
{
public:
    ACommandHandler(AppWindow *mleWindow, IMultiLineEdit *amle);
protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    IMultiLineEdit *mle;
};

/* Create the frame window */
class AppWindow : public IFrameWindow, public ICommandHandler
{
public:
    AppWindow( unsigned long windowId);
    AppWindow();
private:
    about();
    ITitle          title;
    IMultiLineEdit mle;
    ACommandHandler * commandHandler;
};
```


2. Construct the MLE in the .cpp file as follows:

```

/*****
/*          Window Constructor          */
*****/
AppWindow :: AppWindow( unsigned long windowId)
    : IFrameWindow(windowId,          // create Frame window
                    defaultStyle() | menuBar),
      title(this,"MLE Example"),      // include Title
      mle(ID_MLE, this, this)        // create MLE
{
    setClient(&mle);
    handleEventsFor(this);
    setIcon(ID_ICON);
    mle.setFocus();
}
/*****
// Create Command Handler
*****/
commandHandler = new ACommandHandler(this, &mle);
commandHandler->handleEventsFor(this);

```

3. Use the following code for event handling:

```

/*****
/*          Command Handler Constructor          */
*****/
ACommandHandler :: ACommandHandler (AppWindow *mleWindow,
                                     IMultiLineEdit *amle):
    mle(amle)
{
}
/*****
/* MyWindow Command Event Handler          */
*****/
bool ACommandHandler :: command(ICommandEvent& cmdevt)
{
    switch (cmdevt.commandId())
    {
        case ID_IMPORT_ITEM:
            mle->importFromFile("import.txt",IMultiLineEdit:: MLEFormat);
            return true;
        case ID_EXPORT_ITEM:
            mle->exportToFile("export.txt",IMultiLineEdit:: noTran);
            return true;
        case ID_INIT_ITEM:
            mle->setText("This is some initial text.");
            return true;
        case ID_MARK_ITEM:
            mle->selectRange(IRange(13,19));
            return true;
        case ID_COPY_ITEM:
            if (mle->hasSelectedText())
                mle->copy();
            return true;
        case ID_CUT_ITEM:
            if (mle->hasSelectedText())
                mle->cut();
            return true;
        case ID_PASTE_ITEM:
            if (mle->clipboardHasTextFormat())
                mle->paste();
            return true;
        case ID_DELMARK_ITEM:
            if (mle->hasSelectedText())
                mle->discard();
            return true;
        case ID_DELALL_ITEM:
            mle->removeAll();
            return true;
    }
}

```

```

    case ID_INSERT_ITEM:
        mle->add("inserted");
        return true;
    case ID_WORDWRAP_ITEM:
        mle->enableWordWrap(!mle->isWordWrap());
        return true;
    case ID_HOME_ITEM:
        mle->setCursorPosition(0);
        return true;
} /* end switch */
return false;
}

```

Loading and Saving a File

The following member functions from the IMultiLineEdit class allow you to import text to an MLE from a file and export text from an MLE into a file.

Member Function	Use To
importFromFile	Load a file into an MLE
exportToFile	Save from an MLE
exportSelectedTextToFile	Save marked text in an MLE into a file

You can load and save a file to the MLE, as follows:

```

case ID_IMPORT_ITEM:
    mle->importFromFile("import.txt",IMultiLineEdit::MLEFormat);
    return true;
case ID_EXPORT_ITEM:
    mle->exportToFile("export.txt",IMultiLineEdit::noTran);
    return true;

```

Positioning the Cursor

You can position the cursor on a specific line of an MLE or in a specific character position, add to or remove lines from an MLE, or request the number of lines in an MLE.

Position the cursor on the first line, as follows:

```

case ID_HOME_ITEM:
    mle->setCursorPosition(0);
    return true;

```

The following figure, from Windows, shows the cursor on the first line of the MLE:

```

Multi-Line Entry Field Sample
File Edit Options
// .FILE:          amle.cpp
// .DESCRIPTION:   Multiline Edit Sample
//                 Program: Class
//                 Implementation
// .CLASSES:       AEditorWindow
// .IOC WRITER:    Raymond Gallardo
// .IOC WRITER'S DOG: Chester

#include <iapp.hpp>
#include <imgbox.hpp>
#include <ireslib.hpp>
#include <ifiledlg.hpp>
#include <isubmenu.hpp>
#include <ifont.hpp>
#include <ifontdlg.hpp>

```

Performing Clipboard Operations

The `IMultiLineEdit` class has several member functions to perform clipboard operations, including copy, cut, and paste. After you define an MLE, use these member functions to copy text to the clipboard, cut and put text into the clipboard, or paste only the marked lines from the clipboard.

> AIX

In Motif applications, you select a range of text by pressing mouse button 1. To paste text, you use mouse button 2.

The following code performs clipboard operations:

```

case ID_MARK_ITEM: // First mark some text
    mle->selectRange(IRange(13,19));
    return true;
case ID_COPY_ITEM:
    if (mle->hasSelectedText())
        mle->copy();
    return true;
case ID_PASTE_ITEM:
    if (mle->clipboardHasTextFormat())
        mle->paste();
    return true;
case ID_CUT_ITEM: // Check that text is marked
    if (mle->hasSelectedText()) // then cut it to the clipboard
        mle->cut();
    return true;

```

The following figure shows an example of cutting text to the clipboard. It contains marked lines in the client area and a menu option, **Edit**, with three menu items, including **Cut**. The menu item ID of the **Cut** menu item is `ID_CUT_ITEM`.

```
Multi-Line Entry Field Sample
File Edit Options
// Cut Shift+Delete amle.cpp
// Copy Ctrl+Insert Multiline Edit Sample
// Paste Shift+Insert Program: Class
// Implementation
// .CLASSES: AEditorWindow
// .IOC WRITER: Raymond Gallardo
// .IOC WRITER'S DOG: Chester

#include <iapp.hpp>
#include <msgbox.hpp>
#include <ireslib.hpp>
#include <ifiledlg.hpp>
#include <isubmenu.hpp>
#include <ifont.hpp>
#include <ifontdlg.hpp>

cut marked text to clipboard
```

You can find the Multiline Edit Sample in the samples/ioc/mle directory.

RELATED CONCEPTS

“Chapter 7. Entry Fields and Multiline Edit (MLE) Fields” on page 83

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

Multiline Edit Sample

“Create a Static Text Control” on page 81

“Create an Entry Field” on page 83

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Chapter 8. Buttons

A *button* is a type of control window used to initiate an operation or to set the attributes of an operation. A button can appear alone or with a group of other buttons. When buttons are grouped, you can move from button to button within the group by pressing the Arrow keys. You can also move among groups by pressing the Tab key.

A user can select a button by clicking it with the mouse or by pressing the spacebar when the button has the keyboard focus. In most cases, a button changes its appearance when selected.

A button sends notification messages to its owner window when the user selects it.

There are four main types of buttons which determine how the button looks and behaves:

- Push buttons
- Radio buttons
- Check boxes
- Three-state check boxes

▶ AIX

The AIX platform does not support three-state check boxes.

A radio button, check box, or three-state check box *control* an operation; a push button *initiates* an operation.

Make sure that you put each set of radio buttons, check boxes, or three-state check boxes into a single group. Buttons in a group must also have the same parent window. You can do this by either constructing the first button of the set with the `IWindow::group` style, or calling `IWindow::enableGroup()` after you create the first button of the set.

Push buttons

A *push button* is a rectangular window that contains a text string. Typically, an application uses a push button to let the user start or stop an operation. A push button represents an action that is initiated when a user selects it. You can label it with text, graphics, or both. When a user selects a push button, the action occurs immediately if there is a handler for the generated command event.

Use the `IPushButton` class to create and maintain the push button window. By default, a push button generates an application `ICommandEvent`. You can change the default style by changing the window style value to generate a help event or a system command event. Using system command events is not recommended for portable applications. You cannot set colors for push buttons on the Windows operating system.

For example, in the Hello World - Version 3 user interface sample, the `AHelloWindow::command` member function handles the command events generated by pressing the **Left**, **Center**, and **Right** push buttons in the application. Note, the command events are the same as those used for the corresponding menu items.

Therefore, the command function processing is the same whether you press the push button or select the item on the menu bar.

Radio buttons

A *radio button* is a button control that contains a small circular indicator with a text string to the right of it. Use radio buttons to let your users choose from a small list of mutually exclusive items.

For example, an order form application asking the user to select from Small, Medium, and Large could use radio buttons.

Generally, to deselect a radio button you select another radio button in the same group.

If you have a large list of mutually exclusive choices, try using a single-selection list box instead.

The `IRadioButton` class lets you create and manage the radio button window. The `ISelectHandler` class processes the selection of a radio button. You add the handler to either the radio button or its owner window by calling the handler's `handleEventsFor` function.

Check boxes

A *check box* is a button control that contains a small box with a text string to the right of it. They are similar to radio buttons except that you can select anywhere between none and all of the check boxes in a group.

Each time the user selects a check box, that button's state toggles between selected and unselected.

WIN

OS/2

A check mark symbol appears in the check box to indicate that the user has selected the choice.

AIX

Motif fills in the check box to indicate that the user has selected the choice.

Use a check box to set a choice in a group of choices that are not mutually exclusive.

Check boxes may also toggle application features *on* or *off*. For example, a word processing application might use a check box to let the user turn word wrapping on or off.

If you have a large list of items, try using a multiple-selection list box instead.

The `ICheckBox` class lets you create and maintain a check box. The selection of a check box is processed by using the `ISelectHandler` class. You add the handler to either the check box or its owner window.

WIN

OS/2

Three-state check boxes

Three-state check boxes are similar to check boxes except that three-state check boxes have three possible states: selected, unselected, and *half toned*. The half tone state would indicate that the choice is indeterminable.

For example, word processing applications occasionally use three-state check boxes to describe the attributes of a selected block of text, such as italics:

- If the selected block of text contains no italicized text, then the application would display an *unselected* three-state check box.
- If the block of text contains all italicized text, then the application would display a *selected* three-state check box.
- If the block of text contains a mix of italicized and non-italicized text, then the application would display a *half toned* three-state check box.

The `I3StateCheckBox` class lets you create and maintain a three-state check box. You have to use a combination of the `isSelected` and `isHalfTone` member functions to determine the state of a three-state check box. The following table shows the state of the three-state check box and its relation to the `isSelected` and `isHalfTone` member functions.

Current state	isSelected	isHalfTone
checked	true	false
half tone	true	true
not checked	false	false

▶ AIX

The `I3StateCheckBox` class is not supported on Motif.

You process the selection of a three-state checkbox with the `ISelectHandler` class. You can add the handler to either the three-state check box to its owner window.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

RELATED TASKS

“Create a Push Button”

“Create a Radio Button” on page 103

“Create a Check Box” on page 112

“Create a Three-State Check Box” on page 117

“Task and Samples Cross-Reference Table” on page 445

Create a Push Button

A *push button* is a rectangular window that contains a text string. Typically, an application uses a push button to let the user start or stop an operation. A push button represents an action that is initiated when a user selects it. You can label it with text, graphics, or both. When a user selects a push button, the action occurs immediately if there is a handler for the generated command event.

Use the `IPushButton` class to create and maintain the push button window. By default, a push button generates an application `ICommandEvent`. You can change the default style by changing the window style value to generate a help event or a

system command event. Using system command events is not recommended for portable applications. You cannot set colors for push buttons on the Windows operating system

Refer to the reference documentation for a list of the styles provided for IPushButton and for the IPushButton derived class, IGraphicPushButton.

The Hello World version 4 application defines three push buttons (**Left**, **Center**, and **Right**) in the ahellow4.cpp file.

The command events generated by pressing the **Left**, **Center**, and **Right** push buttons are handled by the AHelloWindow::command member function. Note, the command events are the same as those used for the corresponding menu items. Therefore, the command function processing is the same whether you press the push button or select the item on the menu bar.

1. Optionally, in a resource script file, declare the strings that will appear on the buttons. You declare strings in a resource script file in a string table, as in the following excerpt from ahellow4.rc.

```
#include "ahellow4.h"
WND_MAIN ICON ahellow4.ico
STRINGTABLE
BEGIN
    STR_HELLO    , "Hello, World!!!"
// ...
    STR_LEFTTB   , "Left"
    STR_CENTERB  , "Center"
    STR_RIGHTTB  , "Right"
// ...
END
```

2. Declare the push buttons in a header file. This excerpt from ahellow4.hpp declares three IPushButton objects, leftButton, centerButton, and rightButton, as members of the main window class AHelloWindow (which inherits from IFrameWindow). The AHelloWindow class also has an ISetCanvas object called buttons. The sample uses buttons to arrange the push buttons on the main window.

```
class AHelloWindow : public IFrameWindow
{
    // ...
private:
    IMenuBar
        menuBar;
    IStaticText
        statusLine,
        hello;
    ISetCanvas
        buttons;
    IPushButton
        leftButton,
        centerButton,
        rightButton;
    IInfoArea
        infoArea;
    ACommandHandler
        commandHandler;
    AHelloWindow
        &operator=(const AHelloWindow&);
};
```

3. Construct the push buttons and arrange them on the main window. The excerpt is from the ahellow4.cpp file. The AHelloWindow constructor creates the three buttons with the MI_LEFT, MI_CENTER, and MI_RIGHT window IDs. These IDs are

the same ones used by the menu commands **Left**, **Right**, and **Center**. Thus, any any event handling routines created for these menu commands will be the same as for the push buttons. The `AHelloWindow` constructor creates the three buttons as children of the set canvas buttons.

The `AHelloWindow` constructor calls the `IButton::setText` function to place text strings onto the push buttons. The call to `IWindow::enableTabStop` identifies the window as one the user can move the input focus to using the Tab key.

AIX

This function has no effect on tabbing between windows. The Motif operating environment determines how the input focus is changed via the keyboard.

The calls to `ISetCanvas::setMargin` and `ISetCanvas::setPad` configure the set canvas in such a way so that no spaces remain between the push buttons, or between the button and the edge of the frame window. The call to `IFrameWindow::addExtension` puts the set canvas buttons in a frame extension above the information area:

```
AHelloWindow :: AHelloWindow(const unsigned long windowId)
:IFrameWindow(IFrameWindow::defaultStyle() |
              IFrameWindow::minimizedIcon |
              IFrameWindow::accelerator,
              windowId)
,menuBar(windowId, this)
,statusLine(WND_STATUS, this, this)
,hello(WND_HELLO, this, this)
,buttons(WND_BUTTONS, this, this)
,leftButton(MI_LEFT, &buttons, &buttons)
,centerButton(MI_CENTER, &buttons, &buttons)
,rightButton(MI_RIGHT, &buttons, &buttons)
,infoArea(this)
,commandHandler(this)
{
    setClient(&hello);
    addExtension(&statusLine, IFrameWindow::aboveClient,
                IFont(&statusLine).maxCharHeight());
    hello.setText(STR_HELLO);
    leftButton.setText(STR_LEFTB);
    centerButton.setText(STR_CENTERB);
    rightButton.setText(STR_RIGHTB);
    infoArea.setDefaultText(STR_INFO);
    leftButton.enableTabStop();
    buttons.setMargin(ISize());
    buttons.setPad(ISize());
    addExtension(&buttons, IFrameWindow::belowClient,
                (unsigned long)buttons.minimumSize().height());
    commandHandler.handleEventsFor(this);
    setTextAlignment(center);
}
}
```

4. Declare an event handler for the buttons. This is an excerpt from the file `ahellow4.hpp`. The class `ACommandHandler` inherits from `ICommandHandler`.

```
class ACommandHandler : public ICommandHandler
{
public:
    // Constructor
    // - constructs the object with a pointer to the main
    //   frame window
    ACommandHandler(AHelloWindow *helloFrame);
    // Destructor
    virtual
    ~ACommandHandler() { };
protected:
    // Override command function
```

```

// - the command function is called to handle
// application command events
virtual bool
    command(ICommandEvent& cmdEvent);
private:
    AHelloWindow
        *frame;
};

```

5. Override the ICommandEvent::command virtual function so that the application can handle events from the buttons. This function also handles events from the menu as well as events from the keyboard, as defined in the resource script file ahellow4.rc:

```

bool ACommandHandler :: command(ICommandEvent & cmdEvent)
{
    bool eventProcessed(true);
    switch (cmdEvent.commandId()) {
        case MI_CENTER:
            frame->setTextAlignment(AHelloWindow::center);
            break;
        case MI_LEFT:
            frame->setTextAlignment(AHelloWindow::left);
            break;
        case MI_RIGHT:
            frame->setTextAlignment(AHelloWindow::right);
            break;
        case MI_TEXT:
            frame->editText();
            break;
        default:
            eventProcessed=false;
    }
    return(eventProcessed);
}

```

6. Create a command handler, then implement event handling. The constructor of AHelloWindow creates a member object of type ACommandHandler called commandHandler. In the constructor of AHelloWindow the following line of code attaches the command handler to AHelloWindow so it can process event handling:

```

commandHandler.handleEventsFor(this);

```

In the AHelloWindow destructor the following line of code stops event handling:

```

commandHandler.stopHandlingEventsFor(this);

```

The following figure, from Windows, shows the Hello World version 4 push buttons:



RELATED CONCEPTS

- “Chapter 8. Buttons” on page 97
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Hello World Version 4: Adding Dialogs and Push Buttons” on page 420
- “Create a Radio Button”
- “Create a Check Box” on page 112
- “Create a Three-State Check Box” on page 117
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

Create a Radio Button

The following example creates a group of radio buttons. The example creates five radio buttons named **White**, **Black**, **Blue**, **Red** and **Yellow**. An information area indicates which button the user has selected. You can disable or enable the **Blue** radio button by selecting the appropriate command in the menu bar:

1. Define the text associated with each radio button in the resource script file as follows:

```
STRINGTABLE
BEGIN
    STR_BLACK, "Black"
    STR_WHITE, "White"
    STR_BLUE, "Blue"
    STR_RED, "Red"
    STR_YELLOW, "Yellow"
END
```

2. Declare the radio buttons in the header file. This example declares the radio buttons in the main window class called AppWindow:

```
// AppWindow declaration
class AppWindow : public IFrameWindow
{
public:
    AppWindow( unsigned long windowId);
    AppWindow();
    AppWindow & enableButton();
    AppWindow & disableButton();
private:
```

```

        ITitle          *title;
        ISetCanvas     *canvas1;
        IInfoArea      *infoArea;
        IRadioButton   *white;
        IRadioButton   *black;
        IRadioButton   *blue;
        IRadioButton   *red;
        IRadioButton   *yellow;
        IMenuBar        *menuBar;
        MySelectHandler *selectHandler;
        MyCommandHandler *commandHandler;
    };

```

3. Construct the radio buttons in the main window constructor. Because this example adds radio buttons to a set canvas, the example does not have to specify the exact location of the radio button within the main window; the set canvas automatically arranges its controls for you.

The `IWindow::enableGroup()` function groups the five radio buttons. Windows in a *group* are sibling windows consecutively ordered by sibling order. As a result, this example lets you navigate between the radio buttons by using the keyboard arrow keys. Moving forward from the last window in the group causes the input focus to return to the first window in the group. Moving backward from the first window in the group causes the input focus to return to the last window in the group:

```

// AppWindow Constructor
AppWindow :: AppWindow( unsigned long windowId)
    : IFrameWindow(windowId,
        IFrameWindow::defaultStyle())
    {
    // Create Menu Bar
    menuBar = new IMenuBar(windowId, this);
    // Create Title
    title = new ITitle(this, ID_RADIO_TITLE);
    // Create Canvas
    canvas1 = new ISetCanvas(ID_CANVAS, this, this);
    moveSizeTo(IRectangle(0,0,400,100));
    setClient(canvas1);
    IWindow * pParent= canvas1;
    IWindow * pOwner = canvas1;
    // Create Status Area
    infoArea = new IInfoArea(this);
    // Create Radio Buttons
    white = new IRadioButton(WND_WHITEBT, pParent, pOwner);
    white->setText(STR_WHITE);
    black = new IRadioButton(WND_BLACKBT, pParent, pOwner);
    black->setText(STR_BLACK);
    blue = new IRadioButton(WND_BLUEBT, pParent, pOwner);
    blue->setText(STR_BLUE);
    red = new IRadioButton(WND_REDBT, pParent, pOwner);
    red->setText(STR_RED);
    yellow = new IRadioButton(WND_YELLOWBT, pParent, pOwner );
    yellow->setText(STR_YELLOW);
    // Set the group style of the controls
    white->enableGroup();
    // Select white as the default button
    white->select();
    infoArea->setText("White is the currently selected color");
    // Set the select handler to handle events
    selectHandler = new MySelectHandler(infoArea);
    selectHandler->handleEventsFor(white);
    selectHandler->handleEventsFor(black);
    selectHandler->handleEventsFor(blue);
    selectHandler->handleEventsFor(red);
    selectHandler->handleEventsFor(yellow);
    // Set the command handler to handle menu events

```

```

        commandHandler = new MyCommandHandler(this);
        commandHandler->handleEventsFor(this);
        setFocus().show();
    }

```

4. Process events that result from selecting a radio button in the select handler routines. This example, in the header file, declares an event handler called `MySelectHandler` which inherits from `ISelectHandler`. The `MySelectHandler` class overrides the protected function `selected()` so that the example can define what will happen when a user selects a radio button. The `MySelectHandler` class has a pointer to an `IInfoArea` object called `infoArea`. This class uses this pointer to change the text in the information area when a user selects a radio button:

```

// Select handler class declaration
class MySelectHandler: public ISelectHandler
{
public:
    MySelectHandler(IInfoArea *info);
protected:
    selected(IControlEvent& evt);
private:
    bool fProcess;
    IInfoArea *infoArea;
};

```

When a user selects a radio button, an `IControlEvent` object is created. The `MySelectHandler::selected()` function processes the `IControlEvent` object. The `MySelectHandler::selected()` function in this example returns false to indicate that the next event handler should process the selection event (in this example, the selection event is the selection of a radio button):

```

// AppWindow Select Event Handler
MySelectHandler::MySelectHandler(IInfoArea *info)
    :ISelectHandler(),
    infoArea(info)
{
}
// Set static text when radio button selected.
bool MySelectHandler::selected(IControlEvent& evt)
{
    bool fprocess = false;
    switch(evt.controlId())
    {
    case WND_BLACKBT:
        infoArea->setText("Black is the currently selected color");
        fProcess=false;
        break;
    case WND_WHITEBT:
        infoArea->setText("White is the currently selected color");
        fProcess=false;
        break;
    case WND_REDBT:
        infoArea->setText("Red is the currently selected color");
        fProcess=false;
        break;
    case WND_BLUEBT:
        infoArea->setText("Blue is the currently selected color");
        fProcess=false;
        break;
    case WND_YELLOWBT:
        infoArea->setText("Yellow is the currently selected color");
        fProcess=false;
        break;
    }
    return fProcess = false;
}

```

5. Process menu events in the command handler routines. This example, in the header file, declares a command handler called `MyCommandHandler`. This command handler overrides the protected virtual function `ICommandHandler::command()` to process menu commands. The `MyCommandHandler` class stores a pointer to `AppWindow` so that the `MyCommandHandler` can access the member functions and objects of the main window:

```
// Command handler class declaration
class MyCommandHandler : public ICommandHandler {
public:
    MyCommandHandler(AppWindow *mainWindow);
protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    AppWindow *appWindow;
};
```

When a user selects a command from the menu bar, an `ICommandEvent` object is created. The `MyCommandHandler::command()` function processes this `ICommandEvent` object. This function processes two command events: **Disable Blue Button** and **Enable Blue Button**. To process these events, the `MyCommandHandler::command()` function calls `enableButton()` and `disableButton()`, respectively:

```
// Enable the blue radio button
AppWindow & AppWindow::enableButton()
{
    blue->enable();
    return (*this);
}
// Disable the blue radio button
AppWindow & AppWindow::disableButton()
{
    blue->disable();
    return (*this);
}
// Construct the command handler
MyCommandHandler::MyCommandHandler(AppWindow *mainWindow)
{
    appWindow = mainWindow;
}
// AppWindow command event handler
bool MyCommandHandler :: command(ICommandEvent& cmdevt)
{
    switch (cmdevt.commandId())
    {
        case ID_DISABLE_BLUE_BTN:
            appWindow->disableButton();
            break;
        case ID_ENABLE_BLUE_BTN:
            appWindow->enableButton();
            break;
        return true;
    }
    return false;
}
```

The following figure shows the radio buttons created with the preceding code example:



You can find the source to this example in the following files:

- radio.cpp
- radio.h
- radio.hpp
- radio.rc

RELATED CONCEPTS

“Chapter 8. Buttons” on page 97

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“File: radio.cpp”

“File: radio.h” on page 110

“File: radio.hpp” on page 110

“File: radio.rc” on page 111

“Create a Push Button” on page 99

“Create a Check Box” on page 112

“Create a Three-State Check Box” on page 117

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

IRadioButton

Radio Button Example

File: radio.cpp

```
#include <iselhdr.hpp>
#include <ititle.hpp>
#include <iframe.hpp>
#include <iradiobt.hpp>
#include <isetcv.hpp>
#include <icmdhdr.hpp>
#include <iinfoa.hpp>
#include <imenubar.hpp>
#include "radio.h"
#include "radio.hpp"
int main()
{
    if (IBidiSettings::isBidiSupported()) {
        IBidiSettings bd = IBidiSettings::applicationDefaults();
        bd.setWindowLayout(IBidiSettings::layoutRightToLeft);
        IBidiSettings::setApplicationDefaults(bd);
    }
    AppWindow mainWindow(WND_MAIN);
    mainWindow.setFocus().show();
    IApplication::current().run();
    return 0;
}
```

```

}
// AppWindow Constructor
AppWindow :: AppWindow( unsigned long windowId)
    : IFrameWindow(windowId,
                    IFrameWindow::defaultStyle())
    {
    // Create Menu Bar
    menuBar = new IMenuBar(windowId, this);
    // Create Title
    title = new ITitle(this, ID_RADIO_TITLE);
    // Create Canvas
    canvas1 = new ISetCanvas(ID_CANVAS, this, this);
    moveSizeTo(IRectangle(0,0,400,100));
    setClient(canvas1);
    IWindow * pParent= canvas1;
    IWindow * pOwner = canvas1;
    // Create Status Area
    infoArea = new IInfoArea(this);
    // Create Radio Buttons
    white = new IRadioButton(WND_WHITEBT, pParent, pOwner);
    white->setText(STR_WHITE);
    black = new IRadioButton(WND_BLACKBT, pParent, pOwner);
    black->setText(STR_BLACK);
    blue = new IRadioButton(WND_BLUEBT, pParent, pOwner);
    blue->setText(STR_BLUE);
    red = new IRadioButton(WND_REDBT, pParent, pOwner);
    red->setText(STR_RED);
    yellow = new IRadioButton(WND_YELLOWBT, pParent, pOwner );
    yellow->setText(STR_YELLOW);
    // Set the group style of the controls
    white->enableGroup();
    // Select white as the default button
    white->select();
    infoArea->setText("White is the currently selected color");
    // Set the select handler to handle events
    selectHandler = new MySelectHandler(infoArea);
    selectHandler->handleEventsFor(white);
    selectHandler->handleEventsFor(black);
    selectHandler->handleEventsFor(blue);
    selectHandler->handleEventsFor(red);
    selectHandler->handleEventsFor(yellow);
    // Set the command handler to handle menu events
    commandHandler = new MyCommandHandler(this);
    commandHandler->handleEventsFor(this);
    setFocus().show();
}
// AppWindow destructor
AppWindow::~ AppWindow()
{
    commandHandler->stopHandlingEventsFor(this);
}
// AppWindow Select Event Handler
MySelectHandler::MySelectHandler(IInfoArea *info)
    : ISelectHandler(),
      infoArea(info)
{
}
// Set static text when radio button selected.
bool MySelectHandler::selected(IControlEvent& evt)
{
    bool fprocess = false;
    switch(evt.controlId())
    {
    case WND_BLACKBT:
        infoArea->setText("Black is the currently selected color");
        fProcess=false;
        break;

```



```

    case WND_WHITEBT:
        infoArea->setText("White is the currently selected color");
        fProcess=false;
        break;
    case WND_REDBT:
        infoArea->setText("Red is the currently selected color");
        fProcess=false;
        break;
    case WND_BLUEBT:
        infoArea->setText("Blue is the currently selected color");
        fProcess=false;
        break;
    case WND_YELLOWBT:
        infoArea->setText("Yellow is the currently selected color");
        fProcess=false;
        break;
    }
    return fProcess = true;
}
// Enable the blue radio button
AppWindow & AppWindow::enableButton()
{
    blue->enable();
    return (*this);
}
// Disable the blue radio button
AppWindow & AppWindow::disableButton()
{
    blue->disable();
    return (*this);
}
// Construct the command handler
MyCommandHandler::MyCommandHandler(AppWindow *mainWindow)
{
    appWindow = mainWindow;
}
// AppWindow command event handler
bool MyCommandHandler :: command(ICommandEvent& cmdevt)
{
    switch (cmdevt.commandId())
    {
        case ID_DISABLE_BLUE_BTN:
            appWindow->disableButton();
            break;
        case ID_ENABLE_BLUE_BTN:
            appWindow->enableButton();
            break;
        return true;
    }
    return false;
}

```

RELATED CONCEPTS

“Chapter 8. Buttons” on page 97
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Radio Button” on page 103
 “File: radio.h” on page 110
 “File: radio.hpp” on page 110
 “File: radio.rc” on page 111

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: radio.h

```
#define WND_MAIN          100
#define ID_RADIO_TITLE   200
#define ID_CANVAS        210
#define ID_TEXT          220
#define ID_STAT_TITLE    230
#define ID_GROUPBOX      240
#define ID_EDIT          300
#define ID_DISABLE_BLUE_BTN 310
#define ID_ENABLE_BLUE_BTN 320
#define WND_WHITEBT      400
#define WND_BLACKBT      410
#define WND_BLUEBT       420
#define WND_REDBT        430
#define WND_YELLOWBT     440
#define STR_WHITE        500
#define STR_BLACK        510
#define STR_BLUE         520
#define STR_RED          530
#define STR_YELLOW       540
```

RELATED CONCEPTS

“Chapter 8. Buttons” on page 97

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Radio Button” on page 103

“File: radio.cpp” on page 107

“File: radio.hpp”

“File: radio.rc” on page 111

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: radio.hpp

```
// Command handler class declaration
class MyCommandHandler : public ICommandHandler {
public:
    MyCommandHandler(AppWindow *mainWindow);
protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    AppWindow *appWindow;
};
// Select handler class declaration
class MySelectHandler: public ISelectHandler
{
public:
    MySelectHandler(IInfoArea *info);
protected:
    selected(IControlEvent& evt);
private:
    bool fProcess;
    IInfoArea *infoArea;
};
// AppWindow declaration
class AppWindow : public IFrameWindow
```

```

{
public:
    AppWindow( unsigned long windowId);
    AppWindow();
    AppWindow & enableButton();
    AppWindow & disableButton();
private:
    ITitle          *title;
    ISetCanvas     *canvas1;
    IInfoArea      *infoArea;
    IRadioButton   *white;
    IRadioButton   *black;
    IRadioButton   *blue;
    IRadioButton   *red;
    IRadioButton   *yellow;
    IMenuBar       *menuBar;
    MySelectHandler *selectHandler;
    MyCommandHandler *commandHandler;
};

```

RELATED CONCEPTS

“Chapter 8. Buttons” on page 97
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Radio Button” on page 103
 “File: radio.cpp” on page 107
 “File: radio.h” on page 110
 “File: radio.rc”
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

File: radio.rc

WIN

This is a Windows resource script file.

```

#include "basic.h"
STRINGTABLE
BEGIN
    ID_RADIO_TITLE, "Radio Button Example"
    ID_STAT_TITLE, "Color Selection"
    STR_BLACK, "Black"
    STR_WHITE, "White"
    STR_BLUE, "Blue"
    STR_RED, "Red"
    STR_YELLOW, "Yellow"
END
WND_MAIN MENUEX
BEGIN
    POPUP "&Edit", ID_EDIT
    BEGIN
        MENUITEM "&Disable Blue Button", ID_DISABLE_BLUE_BTN
        MENUITEM "&Enable Blue Button", ID_ENABLE_BLUE_BTN
    END
END
END

```

RELATED CONCEPTS

“Chapter 8. Buttons” on page 97
 “Chapter 21. Resources” on page 297

“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Radio Button” on page 103
“Convert Application Resources” on page 395
“File: radio.cpp” on page 107
“File: radio.h” on page 110
“File: radio.hpp” on page 110
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

Create a Check Box

Check boxes are similar to radio buttons, except that they can offer *multiple-choice* selection, as well as individual choice.

WIN

OS/2

When a user selects the choice, a check mark symbol appears in the box to indicate that the choice is selected.

AIX

When a user selects the choice, the check box is filled in to indicate that the choice is selected.

By selecting the choice again, the user deselects the check box. Use a check box to set a choice in a group of choices that are not mutually exclusive.

Check boxes also toggle application features *on* or *off*. For example, a word processing application might use a check box to let the user turn word wrapping on or off.

The `ICheckBox` class lets you create and maintain a check box. The selection of a check box is processed by using the `ISelectHandler` class. You add the handler to either the check box or its owner window.

The following example shows you how to create a check box.

1. Create a check box by first making a declaration in the `checkbo1.hpp` file, as follows:

```
/* *****  
/* Set canvas class declaration */  
/* *****  
class MySet : public ISetCanvas  
{  
public:  
    MySet(unsigned long winId, IWindow* pParent);  
private:  
    ICheckBox check1;  
    ICheckBox check2;  
    ICheckBox check3;  
};  
/* *****  
/* Application Window declaration */  
/* *****  
class AppWindow : public IFrameWindow
```

```

{
public:
    AppWindow( unsigned long windowId);
    AppWindow();
private:
    ITitle          title;
    MySet          * pSetCv;
    IMultiCellCanvas canvas;
};

```

2. Construct the frame window:

```

/*****
/* Window Constructor */
*****/
AppWindow :: AppWindow( unsigned long windowId)
    : IFrameWindow(windowId,
        defaultStyle()),
        title(this,PSZ_OBJECT,PSZ_VIEW),
        canvas(ID_CANVAS, this, this)
{
    setClient(&canvas);
    /*****
    // Create Canvas
    *****/
    /*****
    pSetCv = new MySet(ID_SET, &canvas);
    canvas.addToCell(pSetCv, 2, 2);
}

```

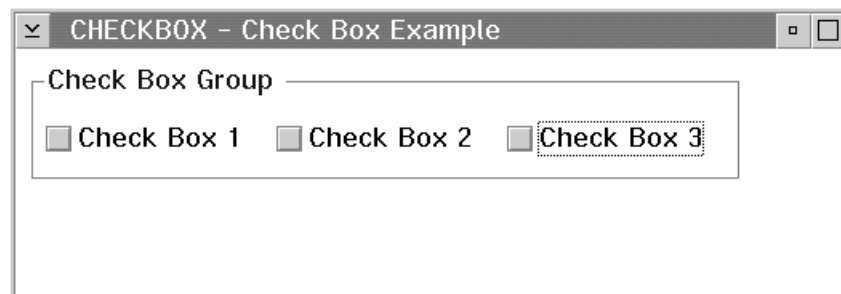
3. Construct the check boxes as shown here:

```

/*****
/* MySet constructor */
*****/
MySet :: MySet(unsigned long winId, IWindow* pParent)
    : ISetCanvas(winId, pParent, pParent),
        check1(ID_BOX1, this, this, IRectangle(),
            ICheckBox::classDefaultStyle |
            IWindow::group),
        check2(ID_BOX2, this, this),
        check3(ID_BOX3, this, this)
{
    check1.setText(PSZ_BOX1);
    check2.setText(PSZ_BOX2);
    check3.setText(PSZ_BOX3);
    setBorderText(IResourceId(PSZ_GROUP));
}

```

The following figure shows the check boxes created using the preceding example:



You can find the complete source to this example in the following files:

- checkbo1.cpp
- checkbo1.h
- checkbo1.hpp

- checkbo1.rc

RELATED CONCEPTS

- “Chapter 8. Buttons” on page 97
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Create a Push Button” on page 99
- “Create a Radio Button” on page 103
- “Create a Three-State Check Box” on page 117
- “File: checkbo1.cpp”
- “File: checkbo1.h” on page 115
- “File: checkbo1.hpp” on page 116
- “File: checkbo1.rc” on page 117
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

Check Box Example

File: checkbo1.cpp

```
#include "checkbo1.h"
#include "checkbo1.hpp"
/*****
 * main
 * - Application entry point.
 *****/
int main()
{
    AppWindow appWindow(ID_MAIN);
    appWindow.setFocus().show();
    IApplication::current().run();
    return 0;
}
/*****
 * Class AppWindow :: AppWindow
 *****/
AppWindow :: AppWindow( unsigned long windowId)
    : IFrameWindow(windowId,
                    defaultStyle()),
      title(this,PSZ_OBJECT,PSZ_VIEW),
      canvas(ID_CANVAS, this, this)
{
    setClient(&canvas);
    // Create Canvas
    pSetCv = new MySet(ID_SET, &canvas);
    canvas.addToCell(pSetCv, 2, 2);
}
/*****
 * Class AppWindow :: ~AppWindow
 *****/
AppWindow :: ~AppWindow ()
{
    delete pSetCv;
}
/*****
 * Class MySet :: MySet
 *****/
MySet :: MySet(unsigned long winId, IWindow* pParent)
    : ISetCanvas(winId, pParent, pParent),
      check1(ID_BOX1, this, this, IRectangle(),
            ICheckBox::classDefaultStyle |
```

```

        IControl::group),
    check2(ID_BOX2, this, this),
    check3(ID_BOX3, this, this)
{
    check1.setText(PSZ_BOX1);
    check2.setText(PSZ_BOX2);
    check3.setText(PSZ_BOX3);
    setBorderText(IResourceId(PSZ_GROUP));
}

```

RELATED CONCEPTS

“Chapter 8. Buttons” on page 97
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Check Box” on page 112
 “File: checkbo1.h”
 “File: checkbo1.hpp” on page 116
 “File: checkbo1.rc” on page 117
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

File: checkbo1.h

```

/*****
 * Window IDs
 *****/
#define ID_ICON 1
#define ID_MAIN 2
#define ID_CANVAS 3
#define ID_SET 4
#define ID_BOX1 5
#define ID_BOX2 6
#define ID_BOX3 7
/*****
 * Menu Bar IDs
 *****/
#define ID_HELP_MENU 900
#define ID_ABOUT_ITEM 910
/*****
 * String table IDs
 *****/
#define PSZ_BOX1 310
#define PSZ_BOX2 320
#define PSZ_BOX3 330
#define PSZ_OBJECT 340
#define PSZ_VIEW 350
#define PSZ_GROUP 360

```

RELATED CONCEPTS

“Chapter 8. Buttons” on page 97
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Check Box” on page 112
 “File: checkbo1.cpp” on page 114
 “File: checkbo1.hpp” on page 116

“File: checkbo1.rc” on page 117

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: checkbo1.hpp

```
#ifndef _CHECKBOX_
#define _CHECKBOX_
#include <iframe.hpp>
#include <ititle.hpp>
#include <icheckbx.hpp>
#include <iapp.hpp>
#include <isetcv.hpp>
#include <imcelcv.hpp>
class AppWindow;
class IFrameWindow;
class ITitle;
class IMultiCellCanvas;
class ISelectHandler;
/*****
 * Class MySet
 *****/
class MySet : public ISetCanvas
{
public:
    MySet(unsigned long winId, IWindow* pParent);
private:
    ICheckBox check1;
    ICheckBox check2;
    ICheckBox check3;
};
/*****
 * Class AppWindow
 *****/
class AppWindow : public IFrameWindow
{
public:
    AppWindow(unsigned long windowId);
    AppWindow();
private:
    ITitle title;
    MySet * pSetCv;
    IMultiCellCanvas canvas;
};
#endif
```

RELATED CONCEPTS

“Chapter 8. Buttons” on page 97

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Check Box” on page 112

“File: checkbo1.cpp” on page 114

“File: checkbo1.h” on page 115

“File: checkbo1.rc” on page 117

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: checkbo1.rc

```
#include "checkbo1.h"
/*****
/* String table
*****/
STRINGTABLE
BEGIN
    PSZ_OBJECT,    "CHECKBOX"
    PSZ_VIEW,     "Check Box Example"
    PSZ_BOX1,     "Check Box 1"
    PSZ_BOX2,     "Check Box 2"
    PSZ_BOX3,     "Check Box 3"
    PSZ_GROUP,    "Check Box Group "
END
```

RELATED CONCEPTS

“Chapter 8. Buttons” on page 97
“Chapter 21. Resources” on page 297
“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Check Box” on page 112
“Convert Application Resources” on page 395
“File: checkbo1.cpp” on page 114
“File: checkbo1.h” on page 115
“File: checkbo1.hpp” on page 116
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

Create a Three-State Check Box

The following example shows how to create a three-state check box:

```
I3StateCheckBox three(ID_THREE, &canvas, &canvas,
    IRectangle(100,220,250,280));
three.setText("3 State");
three.selectHalftone();
// Determine state of the button
if (three.isSelected())
{
    if (three.isHalftone())
    {
        // is halftone
    }
    else
    {
        // is checked
    }
}
else
{
    // is not selected
}
```

RELATED CONCEPTS

“Chapter 8. Buttons” on page 97
“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Push Button” on page 99

“Create a Radio Button” on page 103

“Create a Check Box” on page 112

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

I3StateCheckBox

ISelectHandler

Chapter 9. List Boxes

A *list box* is a control that displays several items at a time, one or more of which can be selected by the user. List boxes are scrollable.

An application uses a list box when it requires a list of selectable fields that is too large for the display area or a list of choices that can change dynamically. Each list item contains a text string and an optional handle. The text string is displayed in the list box window, but the handle is available to the application to reference other data associated with each of the items in the list.

Once you create the list box, your application controls the inserting and deleting of list items. Items can be inserted at the end of the list, automatically sorted into the list, or inserted at a specified index or cursor position. You can add an array of items at a specified index or cursor position.

The `IBaseListBox` class creates and manages list box control windows. Two types of list box controls are derived from `IBaseListBox`. They are `IListBox` and `ICollectionViewListBox`. `IListBox` extends the `IBaseListBox` list box control creation and management to include adding, removing, and replacing list box items. The `ICollectionViewListBox<Element, Collection>` template class extends the `IBaseListBox` control to enable the viewing of an ordered collection as items in a list box. The sequence of elements is the same between the ordered collection and the list box.

You can attach an `ISelectHandler` to a list box or its owner window to process events created when the user selects or double-clicks on an item in the list box.

For an `ICollectionViewListBox` list box, the inserting, sorting, and deleting actions occur on the collection.

You can use cursors to manipulate the list box. Cursors can be filters to process all items in the list box or only the selected ones.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a List Box”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create a List Box

The following example shows you how to create an `IListBox` list box control. The example creates an entry field and a list box. You can type a string into the entry field, then select a menu command to add that string into the list box:

1. In the header file, declare an IListBox object. This example declares an IListBox as a private member of the main window of class Frame. In addition, declare a command handler (called commandHandler in this example) and an event handler (called handleEvent()):

```
class Frame : public IFrameWindow
{
public:
    Frame(unsigned long windowId);
    ~Frame();
    void handleEvent(unsigned long int);
private:
    ITitle title;
    IEntryField ef;
    IListBox listbox;
    IStaticText stTxt1;
    IStaticText stTxt2;
    ACommandHandler * commandHandler;
};
```

2. In the listbox.cpp file, create the IListBox object in the main window constructor. The list box in this example enables multiple selection (the user can select any number of objects at a time in the list box or not select any) and prevents a horizontal scroll bar from appearing within the list box. The frame window positions the listbox in a frame extension with the IFrameWindow::addExtension() function:

```
Frame::Frame(unsigned long windowId)
: IFrameWindow(windowId,
                IFrameWindow::defaultStyle() |
                IFrameWindow::menuBar),
  title(this, "List Box Example"),
  ef(ID_ENTRY, this, this, IRectangle()),
  listbox(ID_LISTBOX, this, this, IRectangle(),
          (ILListBox::defaultStyle()
           | ILListBox::multipleSelect)
          & ILListBox::horizontalScroll),
  stTxt1(ID_STTXT1, this, this),
  stTxt2(ID_STTXT2, this, this)
{
    stTxt1.setAlignment(
        IStaticText::centerLeft);
    addExtension(&stTxt1,
        IFrameWindow::aboveClient);
    addExtension(&ef,
        IFrameWindow::aboveClient);
    stTxt2.setAlignment(
        IStaticText::centerLeft);
    addExtension(&stTxt2,
        IFrameWindow::aboveClient);
    stTxt1.setText("Enter text into entry field:");
    stTxt2.setText("Use the menu to add entry field text to the list box:");
    moveSizeTo(IRectangle(0, 0, 630, 470));
    stTxt1.setBackgroundColor(IColor(199, 212, 204));
    stTxt2.setBackgroundColor(IColor(199, 212, 204));
    ef.setFocus();
    addExtension(&listbox, IFrameWindow::aboveClient);
    show();
    commandHandler = new ACommandHandler(this);
    commandHandler->handleEventsFor(this);
}
```

3. In the listbox.cpp file, add event handling so that you can perform actions on the list box. This example adds event handling with the Frame::handleEvent() function. (The example adds command handling from within the main window constructor.):

```

void Frame :: handleEvent(unsigned long int eventtype)
{
    switch (eventtype)
    {
        case ID_ADD_ITEM:
            if (!ef.isEmpty()) {
                listbox.addAsFirst(ef.text());
                ef.setText("");
            }
            break;
        case ID_ASC_ITEM:
            if (!ef.isEmpty()) {
                listbox.addAscending(ef.text());
                ef.setText("");
            }
            break;
        case ID_DESC_ITEM:
            if (!ef.isEmpty()) {
                listbox.addDescending(ef.text());
                ef.setText("");
            }
            break;
        case ID_DEL_ITEM:
            {
                IListBox::Cursor lbc(listbox, IListBox::Cursor::selectedItems);
                for (lbc.setToFirst(); lbc.isValid(); lbc.setToFirst()) {
                    listbox.removeAt(lbc);
                }
            }
            break;
        case ID_DEALL_ITEM:
            if (!listbox.isEmpty()) {
                listbox.removeAll();
            }
            break;
        case ID_SINGLE_ITEM:
            listbox.disableMultipleSelect();
            listbox.disableExtendedSelect();
            break;
        case ID_MULTI_ITEM:
            listbox.disableExtendedSelect();
            listbox.enableMultipleSelect();
            break;
        case ID_EXTEND_ITEM:
            listbox.disableMultipleSelect();
            listbox.enableExtendedSelect();
            break;
    }
}

```

This event handler lets you perform the following list box commands in this example:

Member function of IListBox	Description
addAsFirst()	Inserts the line of text as the first item in the list box.
addAscending()	Inserts the line of text in ascending sort order.
addDescending()	Inserts the line of text in descending sort order.
removeAt()	Removes the item at the cursor and places the cursor at the next available valid item.
removeAll()	Removes all items from the list box.

Member function of IListBox	Description
Member function of IBaseListBox	
disableMultipleSelect()	Disables multiple selection
disableExtendedSelect()	Disables extended selection
enableMultipleSelect()	Enables <i>multiple selection</i> : the user can select any number of objects at a time in the list box or not select any.
enableExtendedSelect()	Enables <i>extended selection</i> : extends selection to more than one object. This style is a type of selection optimized for the selection of a single object.

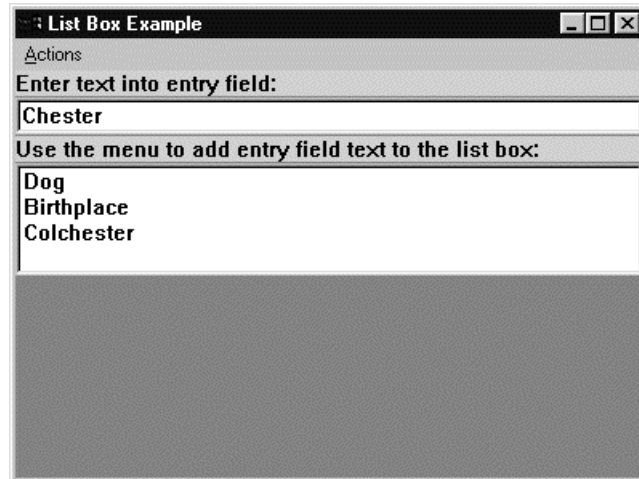
4. Handle commands from the menu bar using a customized command handler. In the constructor for the ACommandHandler class (which is derived from ICommandHandler), make a local copy of the Frame object so its handleEvent function can be called. Then, in the ACommandHandler::command() function, look for the specific items from the menu bar. When application-specific items are found, route them to the Frame::handleEvent() function for processing, as follows:

```

ACommandHandler::ACommandHandler(Frame *listWindow)
{
    list = listWindow;
}
// ...
bool ACommandHandler::command(ICommandEvent& cmdEvent)
{
    switch (cmdEvent.commandId())
    {
        case ID_ADD_ITEM:
        case ID_ASC_ITEM:
        case ID_DESC_ITEM:
        case ID_DEL_ITEM:
        case ID_DELLALL_ITEM:
        case ID_SINGLE_ITEM:
        case ID_MULTI_ITEM:
        case ID_EXTEND_ITEM:
            list->handleEvent(cmdEvent.commandId());
            return true;
    }
    return false;
}

```

The following figure shows the list box created with the previous example. Several elements have been added to the list box:



You can find the complete source to this example in the following files:

- listbox.cpp
- listbox.h
- listbox.hpp
- listbox.rc

RELATED CONCEPTS

“Chapter 9. List Boxes” on page 119

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433

“File: listbox.cpp”

“File: listbox.h” on page 126

“File: listbox.hpp” on page 126

“File: listbox.rc” on page 127

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

IBaseListBox

List Box Example

File: listbox.cpp

```
#include "listbox.h"
#include "listbox.hpp"
#include <icoordsy.hpp>
/*****
* Class Frame :: Frame
* - Window constructor
*****/
Frame::Frame(unsigned long windowId)
    : IFrameWindow(windowId
        , IFrameWindow::defaultStyle()
        | IFrameWindow::menuBar
        | IFrameWindow::accelerator),
```

```

        title(this,"List Box Example"),
        ef(ID_ENTRY,      this, this, IRectangle()),
        listbox(ID_LISTBOX,  this, this, IRectangle()),
                (IListBox::defaultStyle()
                 | IListBox::multipleSelect)
                & IListBox::horizontalScroll),
        stTxt1(ID_STTXT1,   this, this),
        stTxt2(ID_STTXT2,   this, this)
    {
        stTxt1.setAlignment(
            IStaticText::centerLeft);
        addExtension(&stTxt1,
            IFrameWindow::aboveClient);
        addExtension(&ef,
            IFrameWindow::aboveClient);
        stTxt2.setAlignment(
            IStaticText::centerLeft);
        addExtension(&stTxt2,
            IFrameWindow::aboveClient);
        stTxt1.setText("Enter text into entry field:");
        stTxt2.setText("Use the menu to add entry field text to the list box:");
        moveSizeTo(IRectangle(0, 0, 630, 470));
        stTxt1.setBackgroundColor(IColor(199, 212, 204));
        stTxt2.setBackgroundColor(IColor(199, 212, 204));
        ef.setFocus();
        addExtension(&listbox, IFrameWindow::aboveClient);
        show();
        commandHandler = new ACommandHandler(this);
        commandHandler->handleEventsFor(this);
    }
    /*****
    * Class Frame :: ~Frame
    * - Window destructor
    *****/
    Frame::~Frame()
    {
        commandHandler->stopHandlingEventsFor(this);
    }
    /*****
    * Class ACommandHandler :: ACommandHandler
    * - Window destructor
    *****/
    ACommandHandler::ACommandHandler(Frame *listWindow)
    {
        list = listWindow;
    }
    /*****
    * Class Frame :: handleEvent
    *****/
    void Frame :: handleEvent(unsigned long int eventtype)
    {
        switch (eventtype)
        {
            case ID_ADD_ITEM:
                if (!ef.isEmpty()) {
                    listbox.addAsFirst(ef.text());
                    ef.setText("");
                }
                break;
            case ID_ASC_ITEM:
                if (!ef.isEmpty()) {
                    listbox.addAscending(ef.text());
                    ef.setText("");
                }
                break;
            case ID_DESC_ITEM:
                if (!ef.isEmpty()) {

```



```

        listBox.addDescending(ef.text());
        ef.setText("");
    }
    break;
case ID_DEL_ITEM:
    {
        IListBox::Cursor lbc(listBox, IListBox::Cursor::selectedItems);
        for (lbc.setToFirst(); lbc.isValid(); lbc.setToFirst()) {
            listBox.removeAt(lbc);
        }
    }
    break;
case ID_DELALL_ITEM:
    if (!listBox.isEmpty()) {
        listBox.removeAll();
    }
    break;
case ID_SINGLE_ITEM:
    listBox.disableMultipleSelect();
    listBox.disableExtendedSelect();
    break;
case ID_MULTI_ITEM:
    listBox.disableExtendedSelect();
    listBox.enableMultipleSelect();
    break;
case ID_EXTEND_ITEM:
    listBox.disableMultipleSelect();
    listBox.enableExtendedSelect();
    break;
}
}
/*****
* Class ACommandHandler :: command
*****/
bool ACommandHandler::command(ICommandEvent& cmdEvent)
{
    switch (cmdEvent.commandId())
    {
        case ID_ADD_ITEM:
        case ID_ASC_ITEM:
        case ID_DESC_ITEM:
        case ID_DEL_ITEM:
        case ID_DELALL_ITEM:
        case ID_SINGLE_ITEM:
        case ID_MULTI_ITEM:
        case ID_EXTEND_ITEM:
            list->handleEvent(cmdEvent.commandId());
            return true;
    }
    return false;
}
/*****
* main
* - Application entry point.
*****/
int main()
{
    Frame frame(WID_MAIN);
    IApplication::current().run();
    return 0;
}

```

RELATED CONCEPTS

“Chapter 9. List Boxes” on page 119
“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a List Box” on page 119
“File: listbox.h”
“File: listbox.hpp”
“File: listbox.rc” on page 127
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

File: listbox.h

```
#ifndef LISTBOX_H
#define LISTBOX_H
#define WID_MAIN      1000
#define ID_CANVAS     1001
#define ID_ENTRY      1002
#define ID_LISTBOX    1003
#define ID_STTXT1     1004
#define ID_STTXT2     1005
#define ID_ACTION_MENU 2000
#define ID_ADD_ITEM   2001
#define ID_ASC_ITEM   2002
#define ID_DESC_ITEM  2003
#define ID_DEL_ITEM   2004
#define ID_DELALL_ITEM 2005
#define ID_SINGLE_ITEM 2006
#define ID_MULTI_ITEM 2007
#define ID_EXTEND_ITEM 2008
#endif
```

RELATED CONCEPTS

“Chapter 9. List Boxes” on page 119
“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a List Box” on page 119
“File: listbox.cpp” on page 123
“File: listbox.hpp”
“File: listbox.rc” on page 127
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

File: listbox.hpp

```
#include <iframe.hpp>
#include <icmdhdr.hpp>
#include <iapp.hpp>
#include <istattxt.hpp>
#include <icanvas.hpp>
#include <ientryfd.hpp>
#include <ititle.hpp>
#include <ilistbox.hpp>
#include <istring.hpp>
class Frame;
/*****
* Class ACommandHandler *
```

```

*****/
class ACommandHandler : public ICommandHandler
{
public:
    ACommandHandler(Frame *listWindow);
protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    Frame *list;
};
/*****
* Class Frame
*****/
class Frame : public IFrameWindow
{
public:
    Frame(unsigned long windowId);
    Frame();
    void handleEvent(unsigned long int);
private:
    ITitle title;
    IEntryField ef;
    IListBox listbox;
    IStaticText stTxt1;
    IStaticText stTxt2;
    ACommandHandler * commandHandler;
};

```

RELATED CONCEPTS

"Chapter 9. List Boxes" on page 119
 "Chapter 1. Windows" on page 1
 "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a List Box" on page 119
 "File: listbox.cpp" on page 123
 "File: listbox.h" on page 126
 "File: listbox.rc"
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

File: listbox.rc

WIN

The following resource script file is for Windows:

```

#include "listbox.h"
WID_MAIN MENUEX
BEGIN
  POPUP "&Actions" , ID_ACTION_MENU
  BEGIN
    MENUITEM "Add &item\tCtrl+I" , ID_ADD_ITEM
    MENUITEM "Add &ascending\tCtrl+A" , ID_ASC_ITEM
    MENUITEM "Add &descending\tCtrl+D" , ID_DESC_ITEM
    MENUITEM "Delete &selected" , ID_DEL_ITEM
    MENUITEM "Delete a&ll items" , ID_DELLALL_ITEM
    MENUITEM "", 0, MFT_SEPARATOR
    MENUITEM "S&ingle select" , ID_SINGLE_ITEM
    MENUITEM "&Multiple select" , ID_MULTI_ITEM
    MENUITEM "E&xtended select" , ID_EXTEND_ITEM
  END
END

```

```

WID_MAIN ACCELERATORS
BEGIN
    "I", ID_ADD_ITEM , CONTROL
    "A", ID_ASC_ITEM , CONTROL
    "D", ID_DESC_ITEM , CONTROL
END

```

> AIX

> OS/2

The following resource script file is for AIX or OS/2:

```

#include "listbox.h"
MENU WID_MAIN
BEGIN
    SUBMENU "Actions" , ID_ACTION_MENU
    BEGIN
        MENUITEM "Add item\tCtrl+I" , ID_ADD_ITEM
        MENUITEM "Add ascending\tCtrl+A" , ID_ASC_ITEM
        MENUITEM "Add descending\tCtrl+D" , ID_DESC_ITEM
        MENUITEM "Delete selected" , ID_DEL_ITEM
        MENUITEM "Delete all items" , ID_DELLALL_ITEM
        MENUITEM SEPARATOR
        MENUITEM "Single select" , ID_SINGLE_ITEM
        MENUITEM "Multiple select" , ID_MULTI_ITEM
        MENUITEM "Extended select" , ID_EXTEND_ITEM
    END
END
ACCELTABLE WID_MAIN
BEGIN
    "I", ID_ADD_ITEM , CONTROL
    "i", ID_ADD_ITEM , CONTROL
    "A", ID_ASC_ITEM , CONTROL
    "a", ID_ASC_ITEM , CONTROL
    "D", ID_DESC_ITEM , CONTROL
    "d", ID_DESC_ITEM , CONTROL
END

```

RELATED CONCEPTS

["Chapter 9. List Boxes" on page 119](#)
["Chapter 21. Resources" on page 297](#)
["Chapter 1. Windows" on page 1](#)
["Chapter 3. Events and Event Handlers" on page 19](#)

RELATED TASKS

["Create a List Box" on page 119](#)
["Convert Application Resources" on page 395](#)
["File: listbox.cpp" on page 123](#)
["File: listbox.h" on page 126](#)
["File: listbox.hpp" on page 126](#)
["Write an Event Handler" on page 25](#)
["Task and Samples Cross-Reference Table" on page 445](#)

Add or Delete a List Box Item

Your applications can add or delete an item in a list box. Items in a list are specified with a 0-based index (beginning at the top of the list). A new list is created empty; then, the application initializes the list by inserting items.

The application specifies the text and position for each new item. It can specify an absolute-position index or use a list box cursor.

For an `ICollectionViewListBox` control, the list box control window reflects actions on the associated collection. So an element removed from the collection will be visually reflected in the list box control.

The following sample shows you how to create an `IListBox` list box control and then add and delete items. This sample creates two list boxes called `listbox` and `listbox2`. The menu for this sample has five commands:

- **Add items:** Add five elements to `listbox`.
- **Read selected items:** Output the number of selected elements in `listbox2`. Then, for each element selected in `listbox` output the following in `listbox2`:
 - The position of the element in `listbox`
 - The text contained in the element
- **Read all items:** Apply the command **Read selected items** to all elements whether or not an element is selected.
- **Select all:** Selects all the elements in `listbox`.
- **Deselect all:** Deselects all the elements in `listbox`.

You can find the source of the following, from the Listbox Sample in the `samples/ioc/listbox` directory:

1. Declare a frame window with the list box as a child in the `.hpp` file, as follows:

```
class AListBox : public IFrameWindow,
                public ICommandHandler
{
public:
    AListBox( unsigned long windowId );
    AListBox
        &output( const IString& astr ),
        &setStatus();
private:
    IListBox
        listbox,
        listbox2;
    IInfoArea
        infoArea;
    IStaticText
        statusLine;
    IMenuBar
        menuBar;
    ACommandHandler
        commandhandler;
};
```

2. Construct the frame window, initializing the child controls. In this example, the frame window constructor adds the list boxes in frame extensions.

```
AListBox :: AListBox(unsigned long windowId)
: IFrameWindow( IFrameWindow::defaultStyle()
    | IFrameWindow::accelerator
    | IFrameWindow::minimizedIcon,
    windowId),
  listbox(WND_HELLO, this, this, IRectangle(), //Create listbox
    (IListBox::defaultStyle()
    | IListBox::multipleSelect)
    & IListBox::horizontalScroll),
  listbox2(WND_HELLO, this, this, IRectangle(), //Create trace listbox
    IListBox::defaultStyle()
    & IListBox::horizontalScroll),
  infoArea(this),
```

```

        statusLine(WND_STATUS, this, this),
        menuBar(WND_MAIN, this),
        commandhandler(this,&listbox)
    {
        statusLine.setAlignment(
            IStaticText::centerLeft);
        addExtension(&statusLine,
            IFrameWindow::aboveClient, 30UL);
        setStatus();
        addExtension(&listbox,
            IFrameWindow::leftOfClient, 0.5);
        setClient(&listbox2);
        commandhandler.handleEventsFor(this);
        sizeTo(ISize(400,300));
        update();
        setFocus();
        show();
    }

```

3. Handle commands from the menu bar using a customized command handler. In the constructor for the ACommandHandler class (which is derived from ICommandHandler), make a local copy of the AListBox and the IListBox objects so that the command handler can call member functions of AListBox and IListBox. Then, in the ACommandHandler::command() function, look for the specific items from the menu bar. When application-specific items are found, route them to the appropriate function for processing (such as AListBox::output() and AListBox::setStatus()). The AListBox::output() function adds text to the listbox2 list box. The AListBox::setStatus() function creates the text string to put into the listbox2 list box.

```

/*****
 * class AListBox :: output - output status data to listbox2
 *****/
AListBox& AListBox :: output( const IString& astr )
{
    /*-----|
    | Add the string to listbox2
    | If the listbox has more than 7 lines, scroll
    |-----*/
    listbox2.addAsLast( astr );
    if ( listbox2.count() > 7 )
        listbox2.setTop( listbox2.count()-7 );
    return *this;
}
/*****
 * class AListBox :: setStatus - set the status line text
 *****/
AListBox& AListBox :: setStatus()
{
    /*-----|
    | Get the resource library
    | Load a string from the resource library
    | Concatenate the count to the string
    | Set the infoarea text to the string
    |-----*/
    IResourceLibrary reslib;
    IString str=reslib.loadString(STR_INFO);
    str += listbox.count();
    statusLine.setText(str);
    return *this;
}
/*****
 * class ACommandHandler::ACommandHandler - Constructor for the command handler
 *
 * Stores pointers to the main window and the listbox.
 *****/

```

```

*****/
ACommandHandler::ACommandHandler(AListBox *alb,ILListBox *lb1)
    : alistbox(alb)
    , listbox1(lb1)
{
}
/*****
 * ALListBox :: command - command handler
 *****/
bool ACommandHandler :: command(ICommandEvent & cmdEvent)
{
    IResourceLibrary rlib;
    bool fProcessed = true;
    switch (cmdEvent.commandId()) {
        case MI_SELECT_ALL:
            /*-----|
            | Select all the items in listbox1 |
            -----*/
                listbox1->selectAll();
                break;
        case MI_DESELECT_ALL:
            /*-----|
            | Deselect all the items in listbox1 |
            -----*/
                listbox1->deselectAll();
                break;
        case MI_ADD_ITEMS:
            /*-----|
            | Add string items from the resource library to listbox1 |
            | Update the status |
            -----*/
                listbox1->addAsLast(STR_LIST_ITEM1);
                listbox1->addAsLast(STR_LIST_ITEM2);
                listbox1->addAsLast(STR_LIST_ITEM3);
                listbox1->addAsLast(STR_LIST_ITEM4);
                listbox1->addAsLast(STR_LIST_ITEM5);
                alistbox->setStatus();
                break;
        case MI_READ_SEL_ITEMS:
            {
                /*-----|
                | Create a listbox cursor for selected items |
                | Output each line to alistbox |
                | Output the number of selected items |
                -----*/
                    ILListBox::Cursor lbCursor( *listbox1 );
                    for ( lbCursor.setToFirst(); lbCursor.isValid(); lbCursor.setToNext() )
                    {
                        alistbox->output( IString(
                            IString( lbCursor.asIndex() )
                            + " - "
                            + listbox1->elementAt( lbCursor ) ));
                    }
                    alistbox->output( IString( rlib.loadString( STR_SELECTEDITEMS )
                        + IString( listbox1->numberOfSelections() ) ));
                    break;
            }
        case MI_READ_ALL_ITEMS:
            {
                /*-----|
                | Create a listbox cursor for selected items |
                | Output each line to alistbox |
                -----*/
                    ILListBox::Cursor lbCursor( *listbox1, ILListBox::Cursor::allItems );
                    for ( lbCursor.setToFirst(); lbCursor.isValid(); lbCursor.setToNext() )
                    {
                        alistbox->output( IString(

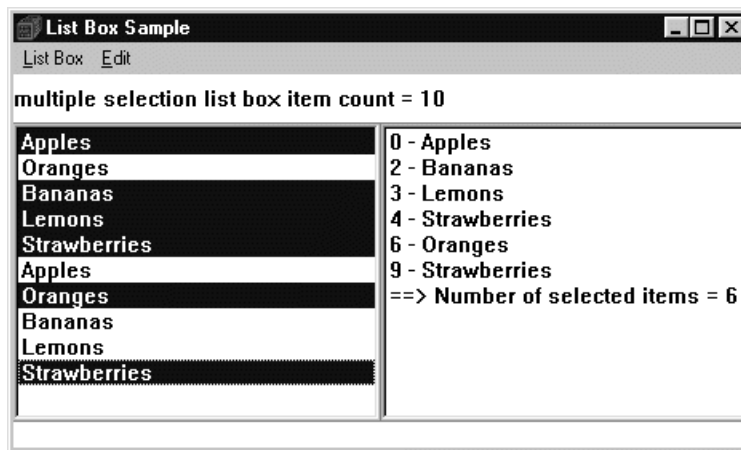
```

```

        IString( lbCursor.asIndex() )
        + " - "
        + listBox1->elementAt( lbCursor ));
    }
    alistbox->output( IString( rlib.loadString( STR_NUMBERITEMS )
        + IString( listBox1->count() )));
    break;
}
default:
    fProcessed = false;           // event not processed
    break;
} /* end switch */
return fProcessed;
}

```

The following figure shows the list box created with the preceding code example. Some items have been added, and the **Read selected items** command has been executed:



You can find the Listbox Sample in the samples/ioc/listbox directory.

RELATED CONCEPTS

- “Chapter 9. List Boxes” on page 119
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- Listbox Sample
- “Create a List Box” on page 119
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

Chapter 10. Combination Boxes

A *combination box*, also called a *prompted entry field*, is two controls in one: an entry field and a list box. There are three types of combination box controls:

- Drop-down: Allows the user to type a choice or to select a choice from a drop-down list.
- Read-only drop-down: Allows the user to select a choice from a drop-down list.
- Simple: Allows the user to type a choice or to select a choice from a list that is always visible.

Drop-down and Read-only drop down boxes allow the user to show and hide the list box when needed.

A combination box control automatically manages the interaction between the entry field and the list box. For example, when the user chooses an item in the list box, the combination box control displays the text for that item in the entry field. Then, the user can edit the text without affecting the item in the list box. When the user types letters in the entry field, the combination box control scrolls the list box contents so that items with those letters become visible

Objects of the `IBaseComboBox` class create and manage combination box control windows. There are two types of combination boxes derived from `IBaseComboBox`: `IComboBox` and `ICollectionViewComboBox`.

The `IComboBox` populates the combination box with the following member functions:

- `IComboBox::add`
- `IComboBox::remove`
- `IComboBox::replaceAt`

The `ICollectionViewComboBox` populates the combination box list box from collection elements using the `setItems` member function. Only `ICollectionViewComboBox` displays its contents from a collection and reflects updates to the collection.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Combination Box”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create a Combination Box

This section shows you how to create a combination box control. The code comes from the Hello World version 6 sample application. The `adialog6.cpp` file does the following:

- Creates a drop-down combination box using the following initializer in the `ATextDialog` constructor:

```
,textField( DID_ENTRY,&clientCanvas,&clientCanvas  
,IRectangle(), IWindow::visible|IComboBox::dropDownType)
```

- Uses a loop to load strings from the resource file into the combination box in ascending order, as follows:

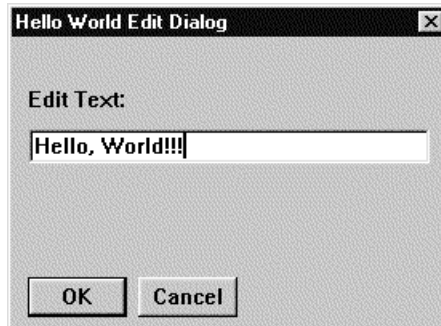
```
for (int i=0;i<HI_COUNT;i++ )  
    textField.addAscending(HI_WORLD+i);
```
- Loads the entry field portion of the combination box with the text string passed into the constructor, using the following code:

```
textField.setText(saveText);
```
- Disables the entry field portion of the combination box for automatic scrolling, adds a margin, and sets a tab stop:

```
textField.disableAutoScroll().enableMargin().enableTabStop();
```
- Retrieves the text that users leave in the entry field portion of the combination box from the combination box object by using the text function, as follows:

```
saveText = textField.text();
```

The following shows the Hello World version 6 combination box control:



RELATED CONCEPTS

“Chapter 10. Combination Boxes” on page 133

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a List Box” on page 119

“Add or Delete a List Box Item” on page 128

“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

IWindow



Chapter 11. Sliders

A *slider* is a visual component that enables a user to set, display, or modify a value by moving the slider arm along the slider shaft. A slider consists of a slider arm, one or two slider scales and, optionally, detents, tick marks, tick text, and slider buttons.



Note that you can have two native slider scales, but only the primary one is visible.

The following table lists the slider components and their descriptions:

Slider Component	Description
  Detent	A user-selectable mark that can be placed anywhere along the slider scale.
Progress indicator	A read-only version of a slider.
Slider arm	An arm that shows the current value by its position on the slider shaft and can be changed programmatically, as well as by users. Users can move the arm along the shaft to set slider values.
Slider buttons	Buttons that move the slider arm incrementally in the indicated direction.
Slider shaft	A track for the slider arm to move along.
Tick text	A label indicating the value the tick mark represents.
Tick mark	An incremental value in a slider scale.

Typically, sliders let users set values that have familiar increments, such as feet, degrees, or decibels. You can use sliders for other purposes when immediate feedback is required, such as to blend colors or show a task's percentage of completion. For example, your application might let a user mix and match color shades by moving a slider arm, or a progress indicator (with the `ribbonStrip` style) could show how much of a task is complete by filling in the slider shaft as the task progresses.

The slider's appearance and user's interaction with a slider is similar to that of a scroll bar. However, these two controls are not interchangeable because each has a unique purpose. A scroll bar scrolls information into view that is outside a window's work area, while the slider sets, displays, or modifies that information.

Your application can specify different scales, sizes, and orientations for its sliders, but the underlying function of the control remains the same.

The ISlider class inherits from the IProgressIndicator class, which is a read-only version of the slider control. Typically, you use a progress indicator to display the percentage of a task that is complete by filling in its shaft as the task progresses. The default for progress indicators is to use the ribbonStrip style to fill the shaft. If you do not use this, there is a slider arm present to indicate the current value. Users cannot move the slider arm in a progress indicator.

You may use the ICircularSlider class to create a circular slider that emulates a dial from typical stereo and video components.

Typically, you use the ISliderArmHandler class to handle events resulting from a user changing a control's input value without releasing the mouse, such as rotating the circular slider or moving the arm of a slider. ISliderArmHandler objects process input tracking events for the ISlider and ICircularSlider controls.

You create a handler derived from ISliderArmHandler and attach it to either the control whose input users can change or to the control's owner window. Call IHandler::handleEventsFor to pass the appropriate control window or owner window to the edit handler.

RELATED CONCEPTS

- "Chapter 9. List Boxes" on page 119
- "Chapter 10. Combination Boxes" on page 133
- "Chapter 11. Sliders" on page 135
- "Chapter 12. Spin Buttons" on page 147
- "Chapter 1. Windows" on page 1
- "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

- "Create a List Box" on page 119
- "Add or Delete a List Box Item" on page 128
- "Create a Combination Box" on page 133
- "Create a Slider Control"
- "Create a Spin Button" on page 147
- "Write an Event Handler" on page 25
- "Task and Samples Cross-Reference Table" on page 445

Create a Slider Control

The following example is comprised of three sliders used to set the red, green, and blue colors in a color mixer. As the slider arm moves, the static text color changes appropriately.

1. Define the main window. A multicell canvas is used as the client window. The client canvas contains the sliders as well as a multicell canvas containing the static text which represents the current color, as follows:

```
class ColorMixerWindow : public IFrameWindow
{
public:
    ColorMixerWindow();
    ~ColorMixerWindow();
    ColorMixerWindow& displayNewColor();
private:
    IMultiCellCanvas canvas;
    ISlider redSlider, greenSlider, blueSlider;
    IMultiCellCanvas colorAreaCanvas;
    ISetCanvas colorAreaFrame;
```

```

        IStaticText      colorArea;
        ColorMonitor     colorMonitor;
        ISetCanvas       redTitleCanvas, greenTitleCanvas, blueTitleCanvas;
        IStaticText      redTitle, greenTitle, blueTitle;
        IStaticText      redValue, greenValue, blueValue;
        IStaticText      mixerTitle;
};

```

2. Define the ColorMonitor class. The ColorMonitor class is used to detect when a slider is moved. The edit member function is overridden to detect when a new color is to be displayed in the color area, as follows:

```

class ColorMonitor : public IEditHandler
{
public:
    ColorMonitor( ColorMixerWindow *colorMixerWindow )
        : _colorMixerWindow( colorMixerWindow )
    {};
protected:
    bool edit( IControlEvent &event );
private:
    ColorMixerWindow *_colorMixerWindow;
};

```

3. Create the main window. Note that the tick spacing defaults to 0 so the slider is automatically sized to the width of the control window:

```

ColorMixerWindow::ColorMixerWindow( )
: IFrameWindow("Slider Example" )
, canvas( ID_MCCANVAS, this, this )
, mixerTitle( ID_MIXER_TITLE, &canvas, &canvas )
, redTitleCanvas( ID_RED_CANVAS, &canvas, &canvas )
, redValue( ID_RED_VALUE, &redTitleCanvas, &redTitleCanvas )
, redTitle( ID_RED_TITLE, &redTitleCanvas, &redTitleCanvas )
, redSlider(ID_RED_SLIDER, &canvas, &canvas, IRectangle(), 256, 0,
    ISlider::pmCompatible |
    ISlider::homeLeft | ISlider::horizontal | ISlider::primaryScale1 |
    ISlider::alignCentered | ISlider::buttonsLeft | IWindow::visible )
, greenTitleCanvas( ID_GREEN_CANVAS, &canvas, &canvas )
, greenValue( ID_GREEN_VALUE, &greenTitleCanvas, &greenTitleCanvas )
, greenTitle( ID_GREEN_TITLE, &greenTitleCanvas, &greenTitleCanvas )
, greenSlider(ID_GREEN_SLIDER, &canvas, &canvas, IRectangle(), 256, 0,
    ISlider::pmCompatible |
    ISlider::homeLeft | ISlider::horizontal | ISlider::primaryScale1 |
    ISlider::alignCentered | ISlider::buttonsLeft | IWindow::visible )
, blueTitleCanvas( ID_BLUE_CANVAS, &canvas, &canvas )
, blueValue( ID_BLUE_VALUE, &blueTitleCanvas, &blueTitleCanvas )
, blueTitle( ID_BLUE_TITLE, &blueTitleCanvas, &blueTitleCanvas )
, blueSlider(ID_BLUE_SLIDER, &canvas, &canvas, IRectangle(), 256, 0,
    ISlider::pmCompatible |
    ISlider::homeLeft | ISlider::horizontal | ISlider::primaryScale1 |
    ISlider::alignCentered | ISlider::buttonsLeft | IWindow::visible )
, colorAreaCanvas( ID_COLOR_CANVAS, &canvas, &canvas )
, colorAreaFrame( ID_COLOR_FRAME, &colorAreaCanvas, &colorAreaCanvas )
, colorArea( ID_COLOR_AREA, &colorAreaCanvas, &colorAreaCanvas )
, colorMonitor( this )
{

```

4. Set up sliders and color area:

```

// Put a border of 10 around the sliders
canvas
    .setColumnWidth(1, 10 )
    .setColumnWidth(5, 10 )
    .setRowHeight( 1, 10 )
    .setRowHeight( 15, 10 )
// Mark the column that contains the sliders as expandable.
    .setColumnWidth(2, 10,true);
mixerTitle
    .setText( "Color Mixer" )

```

```

        .setAlignment( IStaticText::centerCenter );
// Set up the sliders to have a range of 0 to 255 for color selection
redSlider
    .setTickText(0, "0")
    .setTickText(255, "255")
    .moveArmToTick( 0 );
redTitle.setText("Red");
redValue
    .setText("0")
    .setLimit( 3 );
redTitleCanvas
    .setDeckOrientation( ISetCanvas::horizontal )
    .setPackType( ISetCanvas::tight );
greenSlider
    .setTickText(0, "0")
    .setTickText(255, "255")
    .moveArmToTick( 0 );
greenTitle.setText("Green");
greenValue
    .setText("0")
    .setLimit( 3 );
greenTitleCanvas
    .setDeckOrientation( ISetCanvas::horizontal )
    .setPackType( ISetCanvas::tight );
blueSlider
    .setTickText(0, "0")
    .setTickText(255, "255")
    .moveArmToTick( 0 )
    .setForegroundColor(IColor::white );
blueTitle.setText("Blue");
blueValue
    .setText("0")
    .setLimit( 3 );
blueTitleCanvas
    .setDeckOrientation( ISetCanvas::horizontal )
    .setPackType( ISetCanvas::tight );
// Set each slider's background to the color that the slider
// represents.
redSlider.setBackgroundColor ( IColor::red );
greenSlider.setBackgroundColor( IColor::green );
blueSlider.setBackgroundColor ( IColor::blue );
// Add a ISetCanvas to the multicell canvas and use its text feature to
// put a border around the colorArea. The IFont for the colorAreaFrame is
// used to figure out the size of the first row.
IFont colorAreaFont( &colorAreaFrame );
colorAreaCanvas
    .addToCell( &colorAreaFrame, 1, 1, 3, 3 )
    .addToCell( &colorArea, 2, 2 )
    .setColumnWidth( 2, 10, true )
    .setRowHeight( 1, 5 + colorAreaFont.maxCharHeight() )
    .setRowHeight( 2, 10, true );
colorAreaFrame.setBorderText("Color Area");
// Set the slider visible ticks to be every 5th one.
for ( int i = 0; i <= 255; i=i+5 )
{
    redSlider.setTickLength ( i, 10 );
    greenSlider.setTickLength( i, 10 );
    blueSlider.setTickLength ( i, 10 );
}
// Add the controls to the multicell canvas.
canvas
    .addToCell( &mixerTitle,      2, 2 )
    .addToCell( &redTitleCanvas,  2, 4 )
    .addToCell( &redSlider,       2, 6 )
    .setRowHeight( 6, 10, true )
    .addToCell( &greenTitleCanvas, 2, 8 )
    .addToCell( &greenSlider,     2, 10)

```

```

        .setRowHeight( 10, 10, true )
        .addToCell( &blueTitleCanvas, 2, 12)
        .addToCell( &blueSlider,      2, 14)
        .setRowHeight( 14, 10, true )
        .addToCell( &colorAreaCanvas, 4, 4, 1, 11);
// Add the colorMonitor to each slider so that we can detect when
// to update the colorArea.
colorMonitor
    .handleEventsFor( &redSlider )
    .handleEventsFor( &blueSlider )
    .handleEventsFor( &greenSlider );
// Initialize the color areas color.
colorArea.setBackgroundColor( IColor( 0, 0, 0 ));
setClient(&canvas);
}

```

5. Change color area to mix of colors specified by the sliders:

```

ColorMixerWindow& ColorMixerWindow::displayNewColor()
{
    // Use the armTickOffset of each of the sliders to create an IColor
    // object to use to set the background color of the colorArea.
    IColor newColor( redSlider.armTickOffset(),
                    greenSlider.armTickOffset(),
                    blueSlider.armTickOffset() );
    colorArea.setBackgroundColor( newColor );
    // Display the value used to create the colorArea's background color.
    redValue.setText ( IString( redSlider.armTickOffset() ) );
    greenValue.setText( IString( greenSlider.armTickOffset() ) );
    blueValue.setText ( IString( blueSlider.armTickOffset() ) );
    return *this;
}

```

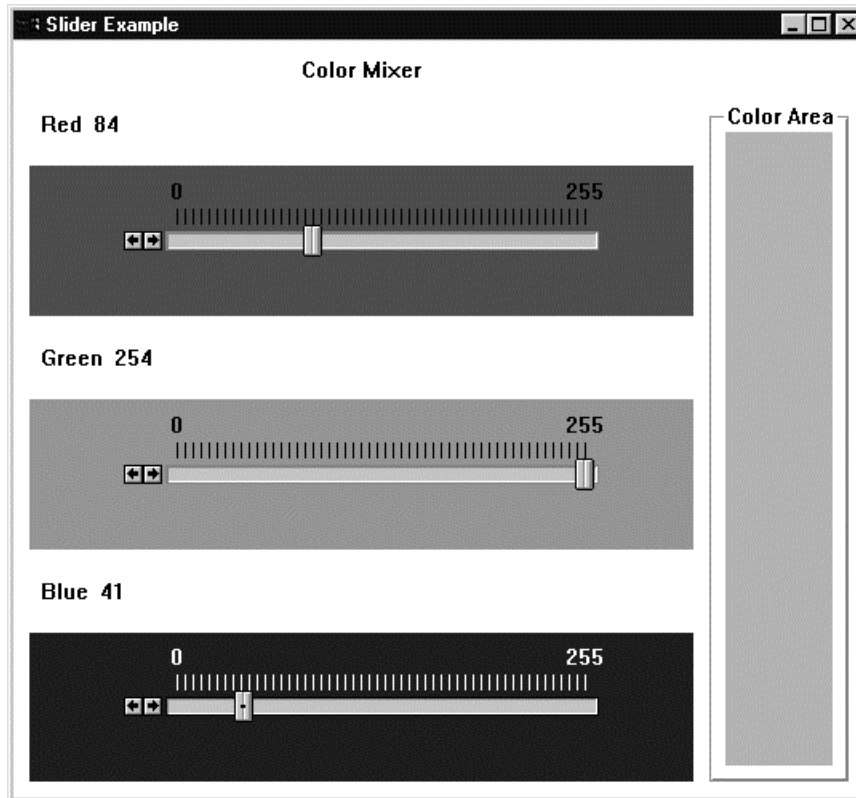
6. Handle the events occurring when a slider's value changes:

```

bool ColorMonitor::edit( IControlEvent& event )
{
    // When the slider's value changes display
    // a new background color in the colorArea.
    // [The color specified in the figure below is
    // the favorite color of G.M.]
    _colorMixerWindow->displayNewColor();
    return true;
}

```

The following figure shows the slider created by the preceding example:



You can find the complete source to this example in the following files:

- slider.cpp
- slider.h
- slider.hpp

RELATED CONCEPTS

"Chapter 11. Sliders" on page 135

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"File: slider.cpp"

"File: slider.h" on page 144

"File: slider.hpp" on page 144

"Write an Event Handler" on page 25

"Task and Samples Cross-Reference Table" on page 445

RELATED REFERENCES

ISlider

Slider Example

File: slider.cpp

```
#include <iapp.hpp>
#include <istring.hpp>
#include <ifont.hpp>
#include "slider.hpp"
#include "slider.h"
```



```

/*****
* main
* - Application entry point.
*****/
int main()
{
    ColorMixerWindow colorMixer;
    colorMixer.setFocus().sizeTo(ISize(540,500)).show();
    IApplication::current().run();
    return 0;
}
/*****
* Class ColorMixerWindow :: ColorMixerWindow
*****/
ColorMixerWindow::ColorMixerWindow()
: IFrameWindow("Slider Example")
, canvas(ID_MCCANVAS, this, this)
, mixerTitle(ID_MIXER_TITLE, &canvas, &canvas)
, redTitleCanvas(ID_RED_CANVAS, &canvas, &canvas)
, redValue(ID_RED_VALUE, &redTitleCanvas, &redTitleCanvas)
, redTitle(ID_RED_TITLE, &redTitleCanvas, &redTitleCanvas)
, redSlider(ID_RED_SLIDER, &canvas, &canvas,
            IRectangle(), 256, 0,
            ISlider::pmCompatible |
            ISlider::homeLeft |
            ISlider::horizontal |
            ISlider::primaryScale1 |
            ISlider::alignCentered |
            ISlider::buttonsLeft |
            IWindow::visible )
, greenTitleCanvas(ID_GREEN_CANVAS, &canvas, &canvas)
, greenValue(ID_GREEN_VALUE, &greenTitleCanvas,
            &greenTitleCanvas)
, greenTitle(ID_GREEN_TITLE, &greenTitleCanvas,
            &greenTitleCanvas)
, greenSlider(ID_GREEN_SLIDER, &canvas, &canvas,
            IRectangle(), 256, 0,
            ISlider::pmCompatible |
            ISlider::homeLeft |
            ISlider::horizontal |
            ISlider::primaryScale1 |
            ISlider::alignCentered |
            ISlider::buttonsLeft |
            IWindow::visible )
, blueTitleCanvas(ID_BLUE_CANVAS, &canvas, &canvas)
, blueValue(ID_BLUE_VALUE, &blueTitleCanvas,
            &blueTitleCanvas)
, blueTitle(ID_BLUE_TITLE, &blueTitleCanvas,
            &blueTitleCanvas)
, blueSlider(ID_BLUE_SLIDER, &canvas, &canvas,
            IRectangle(), 256, 0,
            ISlider::pmCompatible |
            ISlider::homeLeft |
            ISlider::horizontal |
            ISlider::primaryScale1 |
            ISlider::alignCentered |
            ISlider::buttonsLeft |
            IWindow::visible )
, colorAreaCanvas(ID_COLOR_CANVAS, &canvas, &canvas)
, colorArea(ID_COLOR_AREA, &colorAreaCanvas,
            &colorAreaCanvas)
, colorMonitor( this )
{
    // Put a border of 10 around the sliders
    canvas
        .setColumnWidth(1, 10 )
        .setColumnWidth(5, 10 )

```

```

        .setRowHeight( 1, 10 )
        .setRowHeight( 15, 10 )
        // Mark the column that contains the sliders
        // as expandable.
        .setColumnWidth(2, 10,true);
mixerTitle
    .setText( "Color Mixer" )
    .setAlignment( IStaticText::centerCenter );
// Set up the sliders to have a range of 0 to 255
// for color selection
redSlider
    .setTickText(0, "0")
    .setTickText(255, "255")
    .moveArmToTick( 0 );
redTitle.setText("Red");
redValue
    .setText("0")
    .setLimit( 3 );
redTitleCanvas
    .setDeckOrientation( ISetCanvas::horizontal )
    .setPackType( ISetCanvas::tight );
greenSlider
    .setTickText(0, "0")
    .setTickText(255, "255")
    .moveArmToTick( 0 );
greenTitle.setText("Green");
greenValue
    .setText("0")
    .setLimit( 3 );
greenTitleCanvas
    .setDeckOrientation( ISetCanvas::horizontal )
    .setPackType( ISetCanvas::tight );
blueSlider
    .setTickText(0, "0")
    .setTickText(255, "255")
    .moveArmToTick( 0 )
    .setForegroundColor(IColor::kWhite );
blueTitle.setText("Blue");
blueValue
    .setText("0")
    .setLimit( 3 );
blueTitleCanvas
    .setDeckOrientation( ISetCanvas::horizontal )
    .setPackType( ISetCanvas::tight );
// Set each slider's background to the color
// that the slider represents.
redSlider.setBackgroundColor ( IColor::kRed );
greenSlider.setBackgroundColor( IColor::kGreen );
blueSlider.setBackgroundColor ( IColor::kBlue );
// Add the static text control that will show the user's
// selected color to a canvas that draws a border.
colorAreaCanvas
    .addToCell( &colorArea, 1, 1 )
    .setColumnWidth( 1, 10, true )
    .setRowHeight( 1, 10, true );
colorAreaCanvas.setBorderText("Color Area");
// Set the slider visible ticks to be every 5th one.
for ( int i = 0; i <= 255; i=i+5 )
    {
        redSlider.setTickLength ( i, 10 );
        greenSlider.setTickLength( i, 10 );
        blueSlider.setTickLength ( i, 10 );
    }
// Add the controls to the multicell canvas.
canvas
    .addToCell( &mixerTitle,      2, 2 )
    .addToCell( &redTitleCanvas,  2, 4 )

```

```

        .addToCell( &redSlider,      2, 6 )
        .setRowHeight( 6, 10, true )
        .addToCell( &greenTitleCanvas, 2, 8 )
        .addToCell( &greenSlider,    2, 10)
        .setRowHeight( 10, 10, true )
        .addToCell( &blueTitleCanvas, 2, 12)
        .addToCell( &blueSlider,     2, 14)
        .setRowHeight( 14, 10, true )
        .addToCell( &colorAreaCanvas, 4, 4, 1, 11);
// Add the colorMonitor to each slider so that we
// can detect when to update the colorArea.
colorMonitor
    .handleEventsFor( &redSlider )
    .handleEventsFor( &blueSlider )
    .handleEventsFor( &greenSlider );
// Initialize the color areas color.
colorArea.setBackgroundColor( IColor( 0, 0, 0 ));
setClient(&canvas);
}
/*****
* Class ColorMixerWindow :: ~ColorMixerWindow
*****/
ColorMixerWindow::~ColorMixerWindow( )
{
    colorMonitor
        .stopHandlingEventsFor( &redSlider )
        .stopHandlingEventsFor( &blueSlider )
        .stopHandlingEventsFor( &greenSlider );
}
/*****
* Class ColorMixerWindow :: displayNewColor
*****/
ColorMixerWindow& ColorMixerWindow::displayNewColor()
{
    // Use the armTickOffset of each of the sliders to
    // create an IColor object to use to set the background
    // color of the colorArea.
    IColor newColor( redSlider.armTickOffset(),
                    greenSlider.armTickOffset(),
                    blueSlider.armTickOffset() );
    colorArea.setBackgroundColor( newColor );
    // Display the value used to create the colorArea's
    // background color.
    redValue.setText ( IString( redSlider.armTickOffset() ) );
    greenValue.setText( IString( greenSlider.armTickOffset() ) );
    blueValue.setText ( IString( blueSlider.armTickOffset() ) );
    return *this;
}
/*****
* Class ColorMonitor :: edit
*****/
bool ColorMonitor::edit( IControlEvent& event )
{
    // When the slider's value changes display a new
    // background color in the colorArea.
    _colorMixerWindow->displayNewColor();
    return true;
}

```

RELATED CONCEPTS

“Chapter 11. Sliders” on page 135

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

"Create a Slider Control" on page 136
 "File: slider.h"
 "File: slider.hpp"
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

File: slider.h

```

#define ID_MCCANVAS      200
#define ID_RED_SLIDER   201
#define ID_RED_CANVAS   202
#define ID_RED_TITLE    203
#define ID_RED_VALUE    204
#define ID_GREEN_SLIDER 205
#define ID_GREEN_CANVAS 205
#define ID_GREEN_TITLE  206
#define ID_GREEN_VALUE  207
#define ID_BLUE_SLIDER  208
#define ID_BLUE_CANVAS  209
#define ID_BLUE_TITLE   210
#define ID_BLUE_VALUE   211
#define ID_COLOR_CANVAS 212
#define ID_COLOR_AREA   214
#define ID_MIXER_TITLE  215
  
```

RELATED CONCEPTS

"Chapter 11. Sliders" on page 135
 "Chapter 1. Windows" on page 1
 "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a Slider Control" on page 136
 "File: slider.cpp" on page 140
 "File: slider.hpp"
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

File: slider.hpp

```

#ifndef _COLORMIXER_
#define _COLORMIXER_
#include <islider.hpp>
#include <iframe.hpp>
#include <istattxt.hpp>
#include <imcelcv.hpp>
#include <iedithdr.hpp>
#include <isizehdr.hpp>
#include <isetcv.hpp>
#include "slider.h"
class ColorMixerWindow;
/*****
 * Class ColorMonitor
 * - The ColorMonitor class is used to detect when a slider
 *   is moved. The edit member function is overridden to
 *   detect when a new color is to be displayed in the
 *   color Area.
 *****/
class ColorMonitor : public IEditHandler
{
public:
    ColorMonitor( ColorMixerWindow *colorMixerWindow )
        : colorMixerWindow( colorMixerWindow )
    {;}
  
```

```

protected:
    bool edit( IControlEvent &event );
private:
    ColorMixerWindow *_colorMixerWindow;
};
/*****
* Class ColorMixerWindow
* - The ColorMixerWindow is comprised of 3 sliders to set
*   the red, green and blue components for a color mixer
*   sample. The layout is controlled by a IMultiCellCanvas
*   and the color is displayed using a IStaticText.
*****/
class ColorMixerWindow : public IFrameWindow
{
public:
    ColorMixerWindow();
    ~ColorMixerWindow();
    ColorMixerWindow& displayNewColor();
private:
    IMultiCellCanvas canvas;
    ISlider          redSlider, greenSlider, blueSlider;
    IMultiCellCanvas colorAreaCanvas;
    IStaticText      colorArea;
    ColorMonitor     colorMonitor;
    ISetCanvas       redTitleCanvas, greenTitleCanvas, blueTitleCanvas;
    IStaticText      redTitle, greenTitle, blueTitle;
    IStaticText      redValue, greenValue, blueValue;
    IStaticText      mixerTitle;
};
#endif

```

RELATED CONCEPTS

“Chapter 11. Sliders” on page 135
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Slider Control” on page 136
 “File: slider.cpp” on page 140
 “File: slider.h” on page 144
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

Chapter 12. Spin Buttons

A *spin button* control is a visual component that gives users quick access to a finite set of data by letting them select from a scrollable ring of choices. Because the user can see only one item at a time, a spin button should be used only with data that is intuitively related, such as the months of the year or an alphabetic list of cities or states.

A spin button has one spin field and may or may not have spin arrows. The user would find the spin arrows to the right of the spin field. To cycle through the contents of a spin button, the user may click on the arrows or use the up and down arrow keys. The first spin field in the spin button can contain a list of numbers; the second, a list of months; and the third, a list of years.

Master spin buttons have spin arrows, *servant* spin buttons do not. You may relate master and servant spin buttons so that the user may scroll either the master or servant spin button with the spin arrows of the master. To scroll the servant spin button with the spin arrows, the servant spin button must have the input focus.

For example, in setting a date, you can use one master and two servant spin buttons. You can represent the year with the master spin button, and the day and month with the servant spin buttons.

You can create spin buttons from the classes `INumericSpinButton` and `ITextSpinButton`. These classes derive from `IBaseSpinButton`.

The value in a spin button entry field can be an element in an array of data or within a range of integers, defined by an upper and lower limit. Text spin buttons are spin buttons that use arrays of data. Numeric spin buttons are those that use a range of integers.

Attach a handler derived from `ISpinHandler` to the spin button to capture spin events, such as the user pressing the up or down arrow keys.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Spin Button”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create a Spin Button

The following example shows you how to create three spin buttons to show the three parts of a date (month, day, and year). It demonstrates how to initialize the data of both a text spin button and a numeric spin button. It also shows how to retrieve the value of the spin buttons and show the values in a message box. The

spin buttons are children of the client canvas. The frame also contains a status area and a push button. The push button is added as an extension below the client canvas.

1. Define the main window in the spinbtn.hpp file as follows:

```

/*****
/* Declare the frame window */
*****/
class AppWindow : public IFrameWindow {
public:
    AppWindow(unsigned long windowId);
    AppWindow();
    ITextSpinButton * spinbtn1;
    INumericSpinButton * spinbtn2,
    * spinbtn3;
    IStaticText statusarea;
    IPushButton pushbtn;
    void eventHandle();
private:
    ITitle title;
    ICanvas canvas;
    ACommandHandler * commandHandler

```

2. Create the spin buttons. Add the extensions and the command handler in the spinbtn.cpp file:

```

/*****
/* Create the frame window */
*****/
AppWindow::AppWindow(unsigned long windowId)
: IFrameWindow(windowId,
  IFrameWindow::defaultStyle()),
  title(this, "Spin Button Example"),
  canvas(WID_CANVAS, this, this, IRectangle(5, 5, 410, 460)),
  statusarea(WID_STATUS, this, this),
  pushbtn(WID_BUTTON, this, this)
{
    // Customize the push button with text
    pushbtn.setText("OK");
    //Create month spin button (text type spin button)
    spinbtn1 = new ITextSpinButton(WID_MONTH,&canvas, &canvas,
        IRectangle(10,250,200,350));
    //Create day spin button (numeric type spin button)
    spinbtn2 = new INumericSpinButton(WID_DAY,&canvas, &canvas,
        IRectangle(10,150,200,240 ));
    //Create year spin button (numeric type spin button)
    spinbtn3 = new INumericSpinButton(WID_YEAR,&canvas, &canvas,
        IRectangle(10,50,200,140 ));

```

3. Add data to the spin buttons:

```

const int kArraySize = 12;
const int daySize = 31;
const int yearSize = 2000;
const char* textArray[kArraySize] = { "January", "February", "March",
    "April", "May","June", "July", "August", "September", "October",
    "November", "December" };
// Add month data to spin button
for ( int i = 0; i < kArraySize; i++ )
    spinbtn1->addAsLast( textArray[i]);
// Set range of days to day spin button
spinbtn2->setRange( IRange( 1, daySize ) );
// Set range of years to year spin button (1990 - 2000)
spinbtn3->setRange( IRange( 1990, yearSize ) );
//Add the status area as an extension
statusarea.setText("Select a MONTH, DAY and YEAR:");
setClient(&canvas);
addExtension(&statusarea, IFrameWindow::aboveClient,
    .05, IFrameWindow::thickLine);

```



```

//Add the push button as an extension
addExtension(&pushbtn, IFrameWindow::belowClient,
            .05, IFrameWindow::thickLine);
//Add the command handler
commandHandler = new ACommandHandler(this);
commandHandler->handleEventsFor(this);
}

```

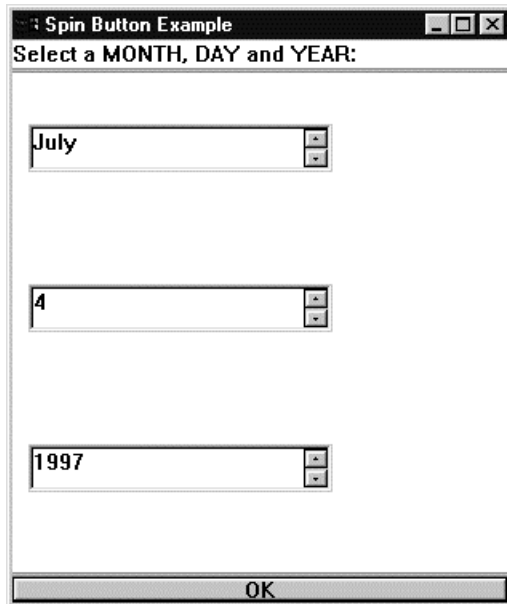
4. Handle an event to show the data in the spin buttons, as follows:

```

void AppWindow :: eventHandle()
{
    IString month = spinbtn1->text() += " ";
    IString day  = spinbtn2->value();
    IString year = spinbtn3->value();
    IString text = "You have selected: ";
    IString date = text += month;
    date += day;
    date += " ";
    date += year;
// Display the data retrieved from the spin buttons
    MessageBox msg(IWindow::desktopWindow());
    msg.setTitle("Spin Buttons Selection Notifier");
    msg.show(date, MessageBox::informationIcon | MessageBox::okButton)
    return;
}
//*****
// ACommandHandler::ACommandHandler
// Construct the command handler from a pointer to the main window
// that events will be handled for.
//*****
ACommandHandler::ACommandHandler(AppWindow *pushWindow)
{
    push = pushWindow;
}
//*****
// ACommandHandler::command
// Handle menu commands
//*****
bool ACommandHandler::command(ICommandEvent & cmdEvent)
{
    bool eventProcessed(true);
// When OK push button is pressed - display the data inside the
// spin buttons
    switch (cmdEvent.commandId())
    {
        case WID_BUTTON:
            push->eventHandle();
            break;
        default:
            eventProcessed = false;
    }
    return true;
}
}

```

The following figure shows the spin buttons created using the preceding example:



You can find the complete source to this example in the following files:

- spinbtn.cpp
- spinbtn.h
- spinbtn.hpp

RELATED CONCEPTS

- “Chapter 12. Spin Buttons” on page 147
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “File: spinbtn.cpp”
- “File: spinbtn.h” on page 152
- “File: spinbtn.hpp” on page 153
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

- ITextSpinButton
- INumericSpinButton

Spin Button Example

File: spinbtn.cpp

```
#include <iostream.h>
#include "spinbtn.h"
#include "spinbtn.hpp"
#include <ifont.hpp>
#include <imgbox.hpp>
#include <icoordsy.hpp>
/*****
* main
* - Application entry point.
*****/
```

```

int main()
{
    ICoordinateSystem::setApplicationOrientation(
        ICoordinateSystem::kOriginLowerLeft);
    AppWindow frameWindow(WID_MAIN);
    frameWindow.setSize(440,400);
    frameWindow.moveSizeToClient(IRectangle(100,100, 410, 410));
    frameWindow.show();
    IApplication::current().run();
    return 0;
}
/*****
* Class AppWindow :: AppWindow
*****/
AppWindow::AppWindow(unsigned long windowId)
: IFrameWindow(windowId,
    IFrameWindow::defaultStyle()),
  title(this, "Spin Button Example"),
  canvas(WID_CANVAS, this, this,
    IRectangle(5, 5, 410, 350)),
  statusarea(WID_STATUS, this, this),
  pushbtn(WID_BUTTON, this, this)
{
    pushbtn.setText("OK");
    // Create month spin button (text type spin button)
    spinbtn1 = new ITextSpinButton(WID_MONTH,&canvas, &canvas,
        IRectangle(10,250,200,280));
    // Create day spin button (numeric type spin button)
    spinbtn2 = new INumericSpinButton(WID_DAY,&canvas, &canvas,
        IRectangle(10,150,200,180 ));
    // Create year spin button (numeric type button)
    spinbtn3 = new INumericSpinButton(WID_YEAR,&canvas, &canvas,
        IRectangle(10,50,200,80 ));

    const int kArraySize = 12;
    const int daySize = 31;
    const int yearSize = 2000;
    const char* textArray[kArraySize] =
        { "January", "February", "March",
          "April", "May", "June", "July",
          "August", "September", "October",
          "November", "December" };
    // Add month data to spin button
    for ( int i = 0; i < kArraySize; i++ )
        spinbtn1->addAsLast( textArray[i]);
    // Set range of days to day spin button
    spinbtn2->setRange( IRange( 1, daySize ) );
    // Set range of years to year spin button (1990 - 2000)
    spinbtn3->setRange( IRange( 1990, yearSize ) );
    statusarea.setText("Select a MONTH, DAY and YEAR:");
    setClient(&canvas);
    addExtension(&statusarea, IFrameWindow::aboveClient,
        .05, IFrameWindow::thickLine);
    addExtension(&pushbtn, IFrameWindow::belowClient,
        .05, IFrameWindow::thickLine);
    commandHandler = new ACommandHandler(this);
    commandHandler->handleEventsFor(this);
}
/*****
* Class AppWindow :: ~AppWindow
*****/
AppWindow::~AppWindow()
{
    commandHandler->stopHandlingEventsFor(this);
    delete commandHandler;
}
/*****
* Class AppWindow :: eventHandle
*****/

```

```

*****/
void AppWindow :: eventHandle()
{
    IString month = spinbtn1->text() += " ";
    IString day   = spinbtn2->value();
    IString year  = spinbtn3->value();
    IString text  = "You have selected: ";
    IString date  = text += month;
    date += day;
    date += " ";
    date += year;
    // Display the data retrieved from the spin buttons
    IMessageBox msg(IWindow::desktopWindow());
    msg.setTitle("Spin Buttons Selection Notifier");
    msg.show(date, IMessageBox::informationIcon | IMessageBox::okButton);
    return;
}
/*****
* Class ACommandHandler :: ACommandHandler *
*****/
ACommandHandler::ACommandHandler(AppWindow *pushWindow)
{
    push = pushWindow;
}
/*****
* Class ACommandHandler :: command *
*****/
bool ACommandHandler::command(ICommandEvent & cmdEvent)
{
    bool eventProcessed(true);
    // Identify the event
    switch (cmdEvent.commandId())
    {
        case WID_BUTTON:
            push->eventHandle();
            break;
        default:
            eventProcessed = false;
    }
    return true;
}

```

RELATED CONCEPTS

“Chapter 12. Spin Buttons” on page 147

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Spin Button” on page 147

“File: spinbtn.h”

“File: spinbtn.hpp” on page 153

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: spinbtn.h

```

#ifndef _SPINBTNH_
#define _SPINBTNH_
#define WID_MAIN 1000
#define WID_CANVAS 1001
#define WID_SPIN 1002
#define WID_MONTH 1003
#define WID_DAY 1004

```

```

#define WID_YEAR    1005
#define WID_BUTTON  1006
#define WID_STATUS  1007
#define WID_ID      1008
#endif

```

RELATED CONCEPTS

“Chapter 12. Spin Buttons” on page 147
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Spin Button” on page 147
 “File: spinbtn.cpp” on page 150
 “File: spinbtn.hpp”
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

File: spinbtn.hpp

```

#ifndef _SPINBTN_
#define _SPINBTN_
#include <ipushbut.hpp>
#include <iframe.hpp>
#include <icmdhdr.hpp>
#include <iapp.hpp>
#include <ititle.hpp>
#include <istattxt.hpp>
#include <icanvas.hpp>
#include <ispinum.hpp>
#include <ispintxt.hpp>
// Forward declarations:
class IFrameWindow;
class AppWindow;
class ICommandHandler;
/*****
 * Class ACommandHandler
 *****/
class ACommandHandler : public ICommandHandler
{
public:
    ACommandHandler(AppWindow *pushWindow);
protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    AppWindow *push;
};
/*****
 * Class AppWindow
 *****/
class AppWindow : public IFrameWindow
{
public:
    AppWindow(unsigned long windowId);
    AppWindow();
    ITextSpinButton * spinbtn1;
    INumericSpinButton * spinbtn2,
    * spinbtn3;
    IStaticText statusarea;
    IPushButton pushbtn;
    void eventHandle();
private:
    ITitle title;

```

```
    ICanvas canvas;  
    ACommandHandler * commandHandler;  
};  
#endif
```

RELATED CONCEPTS

“Chapter 12. Spin Buttons” on page 147

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Spin Button” on page 147

“File: spinbtn.cpp” on page 150

“File: spinbtn.h” on page 152

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Chapter 13. Canvas Controls

A *canvas* is a window that manages its child windows.

Canvases provide portable and flexible alternative to dialog boxes. The Open Class Library does not support dialog templates on AIX, nor does it convert dialog resources between OS/2 and Windows. In addition, the canvas classes give you added features dialog boxes do not provide, including the following:

- Managing the size and position of child windows
- Providing movable split bars between windows
- Supporting the ability to scroll a window

Generally, you build a complex window with a canvas control as the client area. This canvas can contain other canvas controls to build the desired layout.

The canvas classes are as follows:

- ICanvas
 - ISetCanvas
 - ISplitCanvas
 - IMultiCellCanvas
 - IViewPort
 - IDrawingCanvas

The set and multicell canvases automatically size and position their child windows for you, based on the child window's minimum size.

The ICanvas class is the base canvas class. It supports the following basic features:

-

► OS/2

WIN

Dialog-like support on Windows and OS/2 for its child windows:

- Movement from one child window to another with the **Tab** and arrow keys
- Default push buttons
- Mnemonics on buttons, associating a character (sometimes in conjunction with the **Ctrl** or **Alt** key) with a control so that a user may move the input focus to that control by typing the mnemonic character
- Automatic selection of a radio button when users use **Tab** or the arrow keys to give it input focus
- The mutually exclusive selection behavior of buttons: users may select only one radio button from a group
- Protocol for managing the position and size of child windows via the layout and `setLayoutDistorted` virtual functions
- Support for drawing a border with optional text, similar to the static control `IGroupBox`. The border text of a canvas can include font and color information

You must explicitly size and position the child windows of ICanvas objects.

RELATED CONCEPTS

“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Set Canvas” on page 157
“Create a Split Canvas” on page 165
“Create a Multicell Canvas” on page 181
“Create a View Port” on page 193
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

Set Canvases

The `ISetCanvas` class is the simplest canvas class that automatically aligns and sizes its child windows at run-time. You would generally use set canvases to arrange child windows of similar size, such as a group of check boxes or radio buttons, or a row of push buttons.

Decks

A *set canvas* arranges its child windows in either rows or columns. A *deck* is either a row or column of child windows. You can arrange decks oriented horizontally in a column, or conversely, you may arrange decks oriented vertically in a row. By default, the set canvas places all of its child windows in one horizontal deck. If you want to use two or more decks, the set canvas will distribute the child windows as evenly as possible.

The set canvas creates each deck large enough to contain all of its controls.

Child windows may not overlap one another in a set canvas.

Alternatively you may specify the set canvas to create a new deck for each child window with the `group` style. (You would use the style `ISetCanvas::decksByGroup`.) In this case, the set canvas ignores any calls specifying the number of decks.

A set canvas orders its child windows in decks by *sibling order*, the order by which you have created your child windows: left to right for horizontal decks, top to bottom for vertical decks. The **Tab** and arrow keys move the input cursor in the same order.

Pad and Margin

The *pad* is the amount of space the set canvas inserts between child windows. The *margin* is the amount of space the set canvas adds between the edge of the canvas and the child windows. You may modify the default values for pad and margin.

Positioning and Sizing Child Windows

There are three kinds of *pack options* that specify whether the set canvas changes the size and position of its child windows. These options are as follows:

- Tight
 - The set canvas leaves its child windows at their minimum size
 - The set canvas separates its child windows by the default (or specified) pad.
 - This is the default option
- Even
 - The set canvas leaves its child windows at their minimum size

- The set canvas adds more space to the pad such that the child windows appear aligned in rows and columns
- Expanded
 - The canvas makes all child windows the same size according to the maximum width and height among all child windows
 - The set canvas separates its child windows by the default (or specified) pad. As a result, the child windows appear aligned in rows and columns.

You can also use `IAlignmentAttribute` to control the positioning of child windows within a deck.

The set canvas bases the size of each deck on calls to the `minimumSize` member function for each child control. For controls that have sizes defined by the text they contain, such as push buttons and radio buttons, this default processing is normally sufficient. However, for a control that does not have a well-defined size, such as a list box or multiline edit control, you need to set its minimum size by overriding the `calcMinimizeSize` member function or by calling its `setMinimumSize` member function.

A set canvas displays its controls in the same order as declared in the header file.

RELATED CONCEPTS

“Chapter 13. Canvas Controls” on page 155

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Set Canvas”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create a Set Canvas

The following sample shows you how to create a set canvas, as well as how to create a split canvas. The sample uses a split canvas as a client area. The split canvas holds three controls: a static text control and two set canvases. The set canvases each hold seven radio buttons:

1. Declare your canvas objects as member objects of your main window. This code from the header file, `esetcv.hpp`, declares the main window called `ASetCanvas` as a class derived from `IFrameWindow`. It also declares two `ISetCanvas` objects named `vSetCanvas` and `hSetCanvas`:

```
class ASetCanvas : public IFrameWindow
{
public:
    ASetCanvas(unsigned long windowId);
    ~ASetCanvas();
private:
    ISplitCanvas    clientCanvas;
    IStaticText     status;
    ISetCanvas      vSetCanvas,
                  hSetCanvas;
    IRadioButton * radiobut[NUMBER_OF_BUTTONS];
    AButtonHandler buttonHandler;
};
```

2. Construct your main window, filling your set canvases with your controls. The following code from `esetcv.cpp` defines the `ASetCanvas` constructor.

The constructor first creates a split canvas called `clientCanvas`. It then creates three controls as children of `clientCanvas`: `status` (a static text control), `vSetCanvas`, and `hSetCanvas`.

The call to `setDeckOrientation()` specifies the direction of the decks. The call to `setDeckCount()` specifies the maximum number of decks used by the canvas. For example, the code specifies that `vSetCanvas` will orient its decks vertically, and will use a maximum of three decks.

The radio button constructor takes a reference of its parent and owner windows as two of its arguments. In this case, the parent and owner window is one of the set canvases. Note that you do not specify where to place the radio button within the set canvas; set canvases automatically place and arrange their controls for you:

```
ASetCanvas::ASetCanvas(unsigned long windowId)
: IFrameWindow( windowId )
, clientCanvas( WND_SPLITCANVAS, this, this, IRectangle(),
                ISplitCanvas::horizontal | IWindow::visible )
, status( WND_STATUS, &clientCanvas, &clientCanvas )
, vSetCanvas( WND_VSETCANVAS, &clientCanvas, &clientCanvas )
, hSetCanvas( WND_HSETCANVAS, &clientCanvas, &clientCanvas )
{
    // make split canvas the client area
    setClient( &clientCanvas );
    // set alignment of status area text
    status.setAlignment( IStaticText::centerCenter );
    // top canvas has 3 vertical decks
    vSetCanvas.setDeckOrientation( ISetCanvas::vertical );
    vSetCanvas.setDeckCount( 3 );
    // bottom canvas has 3 horizontal decks
    hSetCanvas.setDeckOrientation( ISetCanvas::horizontal );
    hSetCanvas.setDeckCount( 3 );
    // set some space around buttons
    hSetCanvas.setPad( ISize(10,10));
    // give button handler a text control
    buttonHandler.useStatus( &status );
    unsigned int i, mid = (NUMBER_OF_BUTTONS/2);
    // create the first set of radio buttons
    for ( i = 0 ; i < mid ; ++i )
    {
        radiobut[i] = new IRadioButton( WND_BUTTON + i,
                                       &vSetCanvas,
                                       &vSetCanvas );

        // add handler to button
        buttonHandler.handleEventsFor( radiobut[i] );
        radiobut[i]->setText( STR_TEXT + i );
    }
    // set tabStop and Group styles
    radiobut[0]->enableGroup().enableTabStop();
    // select first button in group
    radiobut[0]->select();
    // create the second set of radio buttons
    for ( i = mid ; i < NUMBER_OF_BUTTONS ; ++i )
    {
        radiobut[i] = new IRadioButton( WND_BUTTON + i,
                                       &hSetCanvas,
                                       &hSetCanvas );

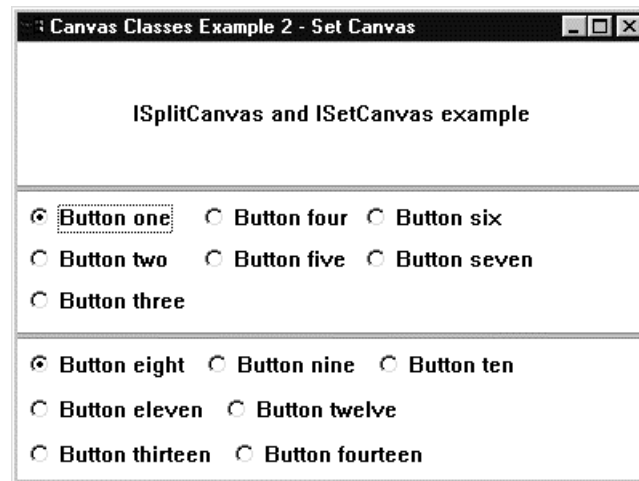
        // add handler to button
        buttonHandler.handleEventsFor( radiobut[i] );
        radiobut[i]->setText( STR_TEXT + i );
    }
    // set tabStop and Group styles
    radiobut[mid]->enableGroup().enableTabStop();
    // select first button in group
    radiobut[mid]->select();
    // set focus to radio button one
}
```

```

radiobut[0]->setFocus();
// set status area text from resource
status.setText( STR_STATUS );
// show main window
show();
}

```

The following figure shows the canvases created with this example:



You can find the complete source to this example in the following files:

- esetcv.cpp
- esetcv.h
- esetcv.hpp
- esetcv.rc

RELATED CONCEPTS

- “Set Canvases” on page 156
- “Chapter 13. Canvas Controls” on page 155
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Create a Split Canvas” on page 165
- “Create a Multicell Canvas” on page 181
- “Create a View Port” on page 193
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

- “File: esetcv.cpp” on page 160
- “File: esetcv.h” on page 162
- “File: esetcv.hpp” on page 162
- “File: esetcv.rc” on page 163
- ISplitCanvas
- IRadioButton

Set Canvas Example

File: esetcv.cpp

```
//
// Set Canvas Example:
// key functions:
// - create a main window
// - run the current application
// - create a horizontal split canvas
// - create static text controls
// - create vertical and horizontal set canvases with
//   multiple decks
// - create radio buttons
// - control cursor and tab movement between different
//   groups
// - process 'Select' events on the radio buttons
// - load strings from resource bound to the exe
//
#include "esetcv.hpp"
/*****
 * main
 * - Application entry point.
 *****/
int main()
{
    ASetCanvas mainWindow(WND_MAIN);
    IApplication::current().run();
    return 0;
}
/*****
 * Class ASetCanvas :: ASetCanvas
 *****/
ASetCanvas::ASetCanvas(unsigned long windowId)
    : IFrameWindow( windowId )
    , clientCanvas( WND_SPLITCANVAS, this, this ,IRectangle(),
                  ISplitCanvas::horizontal | IWindow::visible)
    , status( WND_STATUS, &clientCanvas, &clientCanvas )
    , vSetCanvas( WND_VSETCANVAS, &clientCanvas, &clientCanvas )
    , hSetCanvas( WND_HSETCANVAS, &clientCanvas, &clientCanvas )
{
    // make split canvas the client area
    setClient( &clientCanvas );
    // set alignment of status area text
    status.setAlignment( IStaticText::centerCenter );
    // top canvas has 3 vertical decks
    vSetCanvas.setDeckOrientation( ISetCanvas::vertical );
    vSetCanvas.setDeckCount( 3 );
    // bottom canvas has 3 horizontal decks
    hSetCanvas.setDeckOrientation( ISetCanvas::horizontal );
    hSetCanvas.setDeckCount( 3 );
    // set some space around buttons
    hSetCanvas.setPad( ISize(10,10));
    // give button handler a text control
    buttonHandler.useStatus( &status );
    unsigned int i, mid = (NUMBER_OF_BUTTONS/2);
    // create the first set of radio buttons
    for ( i = 0 ; i < mid ; ++i)
    {
        radiobut[i] = new IRadioButton( WND_BUTTON + i,
                                       &vSetCanvas,
                                       &vSetCanvas );

        // add handler to button
        buttonHandler.handleEventsFor( radiobut[i] );
        radiobut[i]->setText( STR_TEXT + i );
    }
}
```

```

// set tabStop and Group styles
radiobut[0]->enableGroup().enableTabStop();
// select first button in group
radiobut[0]->select();
// create the second set of radio buttons
for ( i = mid ; i < NUMBER_OF_BUTTONS ; ++i)
{
    radiobut[i] = new IRadioButton( WND_BUTTON + i,
                                    &hSetCanvas,
                                    &hSetCanvas );

    // add handler to button
    buttonHandler.handleEventsFor( radiobut[i] );
    radiobut[i]->setText( STR_TEXT + i );
}
// set tabStop and Group styles
radiobut[mid]->enableGroup().enableTabStop();
// select first button in group
radiobut[mid]->select();
// set focus to radio button one
radiobut[0]->setFocus();
// set status area text from resource
status.setText( STR_STATUS );
// show main window
show();
} /* end ASetCanvas :: ASetCanvas(...) */
/*****
* Class ASetCanvas :: ASetCanvas
*****/
ASetCanvas::ASetCanvas()
{
    for (unsigned int i = 0; i < NUMBER_OF_BUTTONS ; ++i)
    {
        delete radiobut[i];
    }
}
/*****
* Class AButtonHandler :: selected
* - an ISelectEventHandler
* - return true if event handled else return false
* - display the number of the button selected in a text
* control
*****/
bool AButtonHandler::selected(IControlEvent& evt )
{
    // assume event is for us
    bool fHandled = true;
    unsigned long ulButtonId = evt.controlId();
    // if the id is one of the buttons
    // then display the button number in the
    // static text control
    if ( ulButtonId >= WND_BUTTON &&
        ulButtonId <= (WND_BUTTON+NUMBER_OF_BUTTONS) &&
        output)
        output->setText( IString( ulButtonId-WND_BUTTON+1 ) );
    else
        // pass event to other handlers
        fHandled = false;
    return fHandled;
} /* end AButtonHandler::selected(...) */

```

RELATED CONCEPTS

“Set Canvases” on page 156

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Create a Set Canvas” on page 157
- “File: esetcv.h”
- “File: esetcv.hpp”
- “File: esetcv.rc” on page 163
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

File: esetcv.h

```
#ifndef ASETCV_H
#define ASETCV_H
#define NUMBER_OF_BUTTONS    14
#define WND_MAIN              0x1000
#define WND_STATUS            0x1001
#define WND_SPLITCANVAS      0x1002
#define WND_VSETCANVAS       0x1003
#define WND_HSETCANVAS       0x1004
#define WND_BUTTON            0x1010
#define STR_STATUS            0x2000
#define STR_VSETCANVAS       0x2001
#define STR_HSETCANVAS       0x2002
#define STR_TEXT              0x2003
#endif
```

RELATED CONCEPTS

- “Set Canvases” on page 156
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Create a Set Canvas” on page 157
- “File: esetcv.cpp” on page 160
- “File: esetcv.hpp”
- “File: esetcv.rc” on page 163
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

File: esetcv.hpp

```
#ifndef ASETCV_HPP
#define ASETCV_HPP
#include <iframe.hpp>
#include <istattxt.hpp>
#include <iradiobt.hpp>
#include <isetcv.hpp>
#include <isplitcv.hpp>
#include <iselhdr.hpp>
#include "esetcv.h"
/*****
 * Class AButtonHandler
 * - This is the select handler for the radio buttons.
 *   The button handler is called when a radio button is
 *   selected. It displays the number of the button in the
 *   static text that is given to it using useInfoArea
 *   member function.
 *****/
class AButtonHandler : public ISelectHandler
{
public:
```

```

        AButtonHandler() : output(0) {}
        useStatus(IStaticText * pstatus)
            { output = pstatus; return 0; }
protected:
    bool
        selected      ( IControlEvent& evt );
private:
    IStaticText      * output;
};
/*****
* Class ASetCanvas
* - Main window for the set canvas example.
* - The window has a split canvas as the client area.
*   Three other controls are added to the split canvas. A
*   static text control is added in the top pane, the lower
*   two panes contain set canvases.
*****/
class ASetCanvas : public IFrameWindow
{
public:
    _ASetCanvas(unsigned long windowId);
    ASetCanvas();
private:
    ISplitCanvas    clientCanvas;
    IStaticText     status;
    ISetCanvas      vSetCanvas,
                    hSetCanvas;
    IRadioButton   * radiobut[NUMBER_OF_BUTTONS];
    AButtonHandler buttonHandler;
};
#endif

```

RELATED CONCEPTS

"Set Canvases" on page 156
 "Chapter 1. Windows" on page 1
 "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a Set Canvas" on page 157
 "File: esetcv.cpp" on page 160
 "File: esetcv.h" on page 162
 "File: esetcv.rc"
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

File: esetcv.rc

```

#include "esetcv.h"
/*****
* string resources
*   Relate symbolic names to literal text strings.
*****/
STRINGTABLE
BEGIN
    WND_MAIN          , "Canvas Classes Example 2 - Set Canvas"
    STR_STATUS        , "ISplitCanvas and ISetCanvas example"
    STR_HSETCANVAS    , "Horizontal Set Canvas"
    STR_VSETCANVAS    , "Vertical Set Canvas"
    (STR_TEXT+0)      , "Button one"
    (STR_TEXT+1)      , "Button two"
    (STR_TEXT+2)      , "Button three"
    (STR_TEXT+3)      , "Button four"
    (STR_TEXT+4)      , "Button five"

```

```

        (STR_TEXT+5)    , "Button six"
        (STR_TEXT+6)    , "Button seven"
        (STR_TEXT+7)    , "Button eight"
        (STR_TEXT+8)    , "Button nine"
        (STR_TEXT+9)    , "Button ten"
        (STR_TEXT+10)   , "Button eleven"
        (STR_TEXT+11)   , "Button twelve"
        (STR_TEXT+12)   , "Button thirteen"
        (STR_TEXT+13)   , "Button fourteen"
    END

```

RELATED CONCEPTS

["Set Canvases" on page 156](#)
["Chapter 21. Resources" on page 297](#)
["Chapter 1. Windows" on page 1](#)
["Chapter 3. Events and Event Handlers" on page 19](#)

RELATED TASKS

["Create a Set Canvas" on page 157](#)
["Convert Application Resources" on page 395](#)
["File: esetcv.cpp" on page 160](#)
["File: esetcv.h" on page 162](#)
["File: esetcv.hpp" on page 162](#)
["Write an Event Handler" on page 25](#)
["Task and Samples Cross-Reference Table" on page 445](#)

Split Canvases

The `ISplitCanvas` class allows you to separate windows with moveable *split bars*. (A split bar behaves in a similar manner to a border of a frame window.)

Panes and Split Bars

A *split canvas* places its child controls into *panes*. Movable or fixed split bars separate these panes. (The default is movable split bars.) Panes manage the position and size of its child windows so that all the child windows (together with the split bars) will fill the entire area of the split canvas.

Child windows may not overlap one another in a split canvas.

You may orient the split bars vertically or horizontally. Moving a split bar changes the sizes of the two windows the split bar separates.

> AIX

You can only move the split canvas using the small square buttons.

> WIN

> OS/2

You can move a split canvas using any part of the split line.

Positioning and Sizing Child Windows

A split canvas will size a child window to entirely fill a pane. Thus, you should use a split canvas to contain controls that you can resize to display more information, such as containers and MLEs. You should not use controls that users

expect to have a certain size or shape. For example, you would not want to put radio buttons in a narrow pane because the pane might clip the text attached to the radio button.

Use the `IListBox::noAdjustPosition` style on a list box control in a split canvas because, otherwise, the operating system may adjust the height of the list box so it will not fill the height of the split canvas.

The order in which you create the child controls determines both their relative position on the split canvas and the order in which tab and cursor keys switch focus between them. For a canvas with vertical split bars, you will find the first child control that you created first in the leftmost pane. For a canvas with horizontal split bars, you will find the first child control in the topmost pane.

RELATED CONCEPTS

“Chapter 13. Canvas Controls” on page 155

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Split Canvas”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create a Split Canvas

The following example shows you how to create a split canvas.

1. Declare a split canvas in the `.hpp` file, as follows:

```
/* ***** */
/* MySplit class - split canvas class and associated */
/* controls */
/* ***** */
class MySplit : public ISplitCanvas {
public:
    MySplit(unsigned long winId, IWindow* pParent);
private:
    IStaticText statTxt1;
    IStaticText statTxt2;
    IStaticText statTxt3;
};
/* ***** */
/* Command handler declaration */
/* ***** */
class ACommandHandler : public ICommandHandler {
public:
    ACommandHandler(AppWindow *asplcan);
protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    AppWindow *splcan;
};
/* ***** */
/* AppWindow class declaration */
/* ***** */
class AppWindow : public IFrameWindow
{
public:
    AppWindow(unsigned long windowId);
    AppWindow();
    AppWindow & updateCanvas(unsigned long eventtype);
};
```

```

private:
    ITitle          *   title;
    MySplit         *   splitCv;
    ACommandHandler *   commandHandler;
};

```

2. Create the window constructor in a .cpp file, as follows:

```

/*****
/* Window constructor */
*****/
AppWindow :: AppWindow( unsigned long windowId)
    : IFrameWindow(windowId,          // create Frame window
                  defaultStyle() | menuBar)
{
/*****
// Create title */
*****/
title = new ITitle(this,PSZ_OBJECT,PSZ_VIEW);
/*****
// Create split canvas */
*****/
splitCv = new MySplit(ID_SPLIT, this);
setClient(splitCv);
/*****
// Create command handler
*****/
commandHandler = new ACommandHandler(this);
commandHandler->handleEventsFor(this);
}
/*****
/*      TestWindow Destructor      */
*****/
AppWindow :: ~AppWindow ()
{
    commandHandler->stopHandlingEventsFor(this);
    delete title;
    delete splitCv;
}
/*****
/*      Perform Menu Events      */
*****/
AppWindow & AppWindow::updateCAN(unsigned long eventtype)
{
    switch (eventtype)
    {
        case ID_VERT_ITEM:
            // Change to vertical orientation
            splitCv->setOrientation(ISplitCanvas::verticalSplit);
            splitCv->refresh();
            return (*this);
        case ID_HORIZ_ITEM:
            // Change to horizontal orientation
            splitCv->setOrientation(ISplitCanvas::horizontalSplit);
            splitCv->refresh();
            return (*this);
        case ID_DOUBLE_EDGE_ITEM:
            // Double the split bar edge
            {
                unsigned long thickness;
                thickness = splitCv->splitBarThickness(ISplitCanvas::splitBarEdge);
                splitCv->setSplitBarThickness(ISplitCanvas::splitBarEdge, thickness*2);
                splitCv->refresh();
                return (*this);
            }
        case ID_HALVE_EDGE_ITEM:
            // Halve the split bar edge
            {

```

```

        unsigned long thickness;
        thickness = splitCv->splitBarThickness(ISplitCanvas::splitBarEdge);
        splitCv->setSplitBarThickness(ISplitCanvas::splitBarEdge, thickness/2);
        splitCv->refresh();
        return (*this);
    }
    case ID_DOUBLE_MIDDLE_ITEM:
        // Double the split bar middle
        {
            unsigned long thickness;
            thickness = splitCv->splitBarThickness(ISplitCanvas::splitBarMiddle);
            splitCv->setSplitBarThickness(ISplitCanvas::splitBarMiddle, thickness*2);
            splitCv->refresh();
            return (*this);
        }
    case ID_HALVE_MIDDLE_ITEM:
        // Halve the split bar middle
        {
            unsigned long thickness;
            thickness = splitCv->splitBarThickness(ISplitCanvas::splitBarMiddle);
            splitCv->setSplitBarThickness(ISplitCanvas::splitBarMiddle, thickness/2);
            splitCv->refresh();
            return (*this);
        }
    } /* end switch */
    return (*this);
}

/*****
/*          Construct the Command Handler          */
*****/
ACommandHandler::ACommandHandler(AppWindow *asplcnv)
{
    splcan = asplcnv;
}

/*****
/*          MyWindow Command Event Handler          */
*****/
bool ACommandHandler :: command(ICommandEvent& cmdEvent)
{
    switch (cmdEvent.commandId())
    {
        case ID_VERT_ITEM:
        case ID_HORIZ_ITEM:
        case ID_DOUBLE_EDGE_ITEM:
        case ID_HALVE_EDGE_ITEM:
        case ID_DOUBLE_MIDDLE_ITEM:
        case ID_HALVE_MIDDLE_ITEM:
            splcan->updateCAN(cmdEvent.commandId());
            break;
            return true;
    }
    return false;
}

/*****
/* MySplit constructor          */
*****/
MySplit :: MySplit(unsigned long winId, IWindow* pParent)
: ISplitCanvas(winId, pParent, pParent),
  statTxt1(ID_TEXT1, this, this),
  statTxt2(ID_TEXT2, this, this),
  statTxt3(ID_TEXT3, this, this)
{
    setSplitWindowPercentage(&statTxt1, 20);
    setSplitWindowPercentage(&statTxt2, 40);
    setSplitWindowPercentage(&statTxt3, 40);
    statTxt1.setText(PSZ_TEXT1);
    statTxt2.setText(PSZ_TEXT2);
}

```

```

statTxt3.setText(PSZ_TEXT3);
statTxt1.setBackgroundColor(IColor(254, 254, 201));
statTxt2.setBackgroundColor(IColor(190, 204, 220));
statTxt3.setBackgroundColor(IColor(199, 212, 204));
}

```

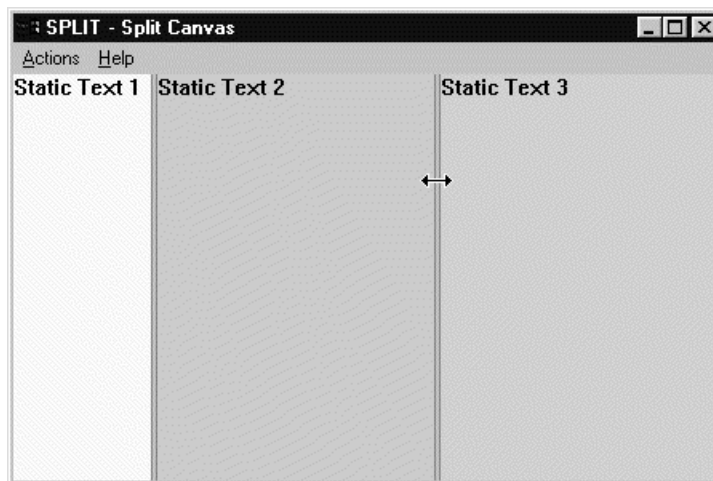
3. Handle command events:

```

/*****
/* Construct the command handler */
/*****
ACommandHandler::ACommandHandler(AppWindow *aspIcnv)
{
    splcan = aspIcnv;
}
/*****
/* MyWindow command event handler */
/*****
bool ACommandHandler :: command(ICommandEvent& cmdEvent)
{
    switch (cmdEvent.commandId())
    {
        case ID_VERT_ITEM:
        case ID_HORIZ_ITEM:
        case ID_DOUBLE_EDGE_ITEM:
        case ID_HALVE_EDGE_ITEM:
        case ID_DOUBLE_MIDDLE_ITEM:
        case ID_HALVE_MIDDLE_ITEM:
            splcan->updateCanvas(cmdEvent.commandId());
            return true;
    }
    return false;
}

```

The following figure shows the split canvas created by using the preceding example:



You can find the complete source to this example in the following files:

- splitcan.cpp
- splitcan.h
- splitcan.hpp
- splitcan.rc

The following examples show how to create a window containing two split canvases. Each pane is occupied by a static text control.

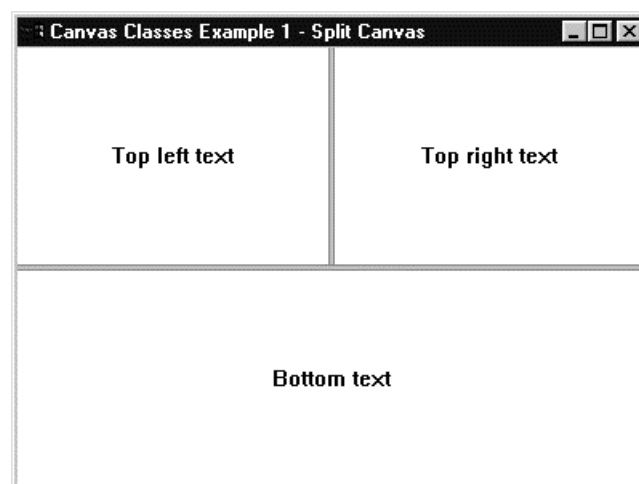
1. This code from the header file declares the ASplitCanvas class as an IFrameWindow derived class. Member functions are initialized in the order that they appear in the class declaration.

```
#include <iframe.hpp> // IFrameWindow
#include <istattxt.hpp> // IStaticText
#include <isplitcv.hpp> // ISplitCanvas class
ASplitCanvas : public IFrameWindow {
public:
    ASplitCanvas(unsigned long windowId); // Constructor
private:
    ISplitCanvas horzCanvas, // The canvases will be created
                    vertCanvas; // in the same order they
    IStaticText lText, // are declared.
                rText,
                bText;
};
```

2. This code creates the window:

```
ASplitCanvas :: ASplitCanvas( unsigned long windowId )
: IFrameWindow( windowId )
, horzCanvas( WND_CANVAS, this, this )
, vertCanvas( WND_CANVAS2, &horzCanvas, &horzCanvas )
, lText( WND_TXTL, &vertCanvas, &vertCanvas )
, rText( WND_TXTR, &vertCanvas, &vertCanvas )
, bText( WND_TXTB, &horzCanvas, &horzCanvas )
{
    // give the canvas a horizontal split bar
    // and make it the client area
    horzCanvas.setOrientation( ISplitCanvas::horizontalSplit );
    setClient( &horzCanvas );
    // give the canvas a vertical split bar
    vertCanvas.setOrientation( ISplitCanvas::verticalSplit );
    // set top left static text
    lText.setText(STR_TOPLEFT);
    lText.setAlignment( IStaticText::centerCenter );
    // set top right static text
    rText.setText(STR_TOPRIGHT);
    rText.setAlignment( IStaticText::centerCenter );
    // set bottom static text
    bText.setText(STR_BOTTOM);
    bText.setAlignment( IStaticText::centerCenter );
    // set focus and show window
    setFocus().show();
}
```

The following figure shows the completed split canvas:



You can find the complete source to this example in the following files:

- esplitcv.cpp
- esplitcv.h
- esplitcv.hpp
- esplitcv.rc

RELATED CONCEPTS

“Split Canvases” on page 164
“Chapter 13. Canvas Controls” on page 155
“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Set Canvas” on page 157
“Create a Multicell Canvas” on page 181
“Create a View Port” on page 193
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

“File: splitcan.cpp”
“File: splitcan.h” on page 173
“File: splitcan.hpp” on page 173
“File: splitcan.rc” on page 175
“File: esetcv.cpp” on page 160
“File: esetcv.h” on page 162
“File: esetcv.hpp” on page 162
“File: esetcv.rc” on page 163
ISplitCanvas

Split Canvas Example 1

File: splitcan.cpp

```
#include "splitcan.h"
#include "splitcan.hpp"
/*****
* main
* - Application entry point.
*****/
int main()
{
    AppWindow appWindow(ID_MAIN);
    appWindow.setFocus().show();
    IApplication::current().run();
    return 0;
}
/*****
* Class AppWindow :: AppWindow
*****/
AppWindow :: AppWindow( unsigned long windowId)
    : IFrameWindow(windowId, defaultStyle() | menuBar)
{
    // Create Title
    title = new ITitle(this,PSZ_OBJECT,PSZ_VIEW);
    // Create SplitCanvas
    splitCv = new MySplit(ID_SPLIT, this);
}
```

```

        setClient(splitCv);
        // Create Command Handler
        commandHandler = new ACommandHandler(this);
        commandHandler->handleEventsFor(this);
    }
    /*****
    * Class AppWindow :: ~AppWindow
    *****/
AppWindow :: ~AppWindow ()
{
    commandHandler->stopHandlingEventsFor(this);
    delete title;
    delete splitCv;
}
/*****
* Class AppWindow :: updateCAN
*****/
AppWindow & AppWindow::updateCAN(unsigned long eventtype)
{
    switch (eventtype)
    {
        case ID_VERT_ITEM:
        {
            // Change to vertical orientation
            splitCv->setOrientation(ISplitCanvas::verticalSplit);
            splitCv->refresh();
            return (*this);
        }
        case ID_HORIZ_ITEM:
        {
            // Change to horizontal orientation
            splitCv->setOrientation(ISplitCanvas::horizontalSplit);
            splitCv->refresh();
            return (*this);
        }
        case ID_DOUBLE_EDGE_ITEM:
        {
            // Double the split bar edge
            unsigned long thickness;
            thickness = splitCv->splitBarThickness(
                ISplitCanvas::splitBarEdge);
            splitCv->setSplitBarThickness(
                ISplitCanvas::splitBarEdge, thickness*2);
            splitCv->refresh();
            return (*this);
        }
        case ID_HALVE_EDGE_ITEM:
        {
            // Halve the split bar edge
            unsigned long thickness;
            thickness = splitCv->splitBarThickness(
                ISplitCanvas::splitBarEdge);
            splitCv->setSplitBarThickness(
                ISplitCanvas::splitBarEdge, thickness/2);
            splitCv->refresh();
            return (*this);
        }
        case ID_DOUBLE_MIDDLE_ITEM:
        {
            // Double the split bar middle
            unsigned long thickness;
            thickness = splitCv->splitBarThickness(
                ISplitCanvas::splitBarMiddle);
            splitCv->setSplitBarThickness(
                ISplitCanvas::splitBarMiddle, thickness*2);
            splitCv->refresh();
            return (*this);
        }
    }
}

```

```

    }
    case ID_HALVE_MIDDLE_ITEM:
    {
        // Halve the split bar middle
        unsigned long thickness;
        thickness = splitCv->splitBarThickness(
            ISplitCanvas::splitBarMiddle);
        splitCv->setSplitBarThickness(
            ISplitCanvas::splitBarMiddle, thickness/2);
        splitCv->refresh();
        return (*this);
    }
}
return (*this);
}
/*****
* Class ACommandHandler :: ACommandHandler
*****/
ACommandHandler::ACommandHandler(AppWindow *asplcnv)
{
    splcan = asplcnv;
}
/*****
* Class ACommandHandler :: command
*****/
bool ACommandHandler :: command(ICommandEvent& cmdEvent)
{
    switch (cmdEvent.commandId())
    {
        case ID_VERT_ITEM:
        case ID_HORIZ_ITEM:
        case ID_DOUBLE_EDGE_ITEM:
        case ID_HALVE_EDGE_ITEM:
        case ID_DOUBLE_MIDDLE_ITEM:
        case ID_HALVE_MIDDLE_ITEM:
            splcan->updateCAN(cmdEvent.commandId());
            break;
            return true;
    }
    return false;
}
/*****
* Class MySplit :: MySplit
* - Create a split canvas with three areas of 20,40 and
*   40%
*****/
MySplit :: MySplit(unsigned long winId, IWindow* pParent)
: ISplitCanvas(winId, pParent, pParent),
  statTxt1(ID_TEXT1, this, this),
  statTxt2(ID_TEXT2, this, this),
  statTxt3(ID_TEXT3, this, this)
{
    setSplitWindowPercentage(&statTxt1, 20);
    setSplitWindowPercentage(&statTxt2, 40);
    setSplitWindowPercentage(&statTxt3, 40);
    statTxt1.setText(PSZ_TEXT1);
    statTxt2.setText(PSZ_TEXT2);
    statTxt3.setText(PSZ_TEXT3);
    statTxt1.setBackgroundColor(IColor(254, 254, 201));
    statTxt2.setBackgroundColor(IColor(190, 204, 220));
    statTxt3.setBackgroundColor(IColor(199, 212, 204));
}

```

RELATED CONCEPTS

“Split Canvases” on page 164

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Split Canvas” on page 165

“File: splitcan.h”

“File: splitcan.hpp”

“File: splitcan.rc” on page 175

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: splitcan.h

```
/*
 * Window and Control IDs
 */
#define ID_MAIN 2
#define ID_SPLIT 3
#define ID_TEXT1 4
#define ID_TEXT2 5
#define ID_TEXT3 6
/*
 * Menu Bar IDs
 */
#define ID_ACTIONS_MENU 100
#define ID_VERT_ITEM 110
#define ID_HORIZ_ITEM 120
#define ID_DOUBLE_EDGE_ITEM 130
#define ID_HALVE_EDGE_ITEM 140
#define ID_DOUBLE_MIDDLE_ITEM 150
#define ID_HALVE_MIDDLE_ITEM 160
#define ID_HELP_MENU 900
#define ID_ABOUT_ITEM 910
/*
 * String table IDs
 */
#define PSZ_OBJECT 300
#define PSZ_VIEW 310
#define PSZ_TEXT1 320
#define PSZ_TEXT2 330
#define PSZ_TEXT3 340
```

RELATED CONCEPTS

“Split Canvases” on page 164

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Split Canvas” on page 165

“File: splitcan.cpp” on page 170

“File: splitcan.hpp”

“File: splitcan.rc” on page 175

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

File: splitcan.hpp

```
#ifndef _SPLITCAN_
#define _SPLITCAN_
#include <iframe.hpp>
#include <icmdhdr.hpp>
#include <ititle.hpp>
#include <iapp.hpp>
```

```

#include <imgbox.hpp>
#include <istattxt.hpp>
#include <isplitcv.hpp>
class ITitle;
class MySplit;
class IFrameWindow;
class ICommandHandler;
class AppWindow;
class IStaticText;
/*****
* Class MySplit
* - Split canvas class and associated controls
*****/
class MySplit : public ISplitCanvas
{
public:
    MySplit(unsigned long winId, IWindow* pParent);
private:
    IStaticText statTxt1;
    IStaticText statTxt2;
    IStaticText statTxt3;
};
/*****
* Class ACommandHandler
*****/
class ACommandHandler : public ICommandHandler
{
public:
    ACommandHandler(AppWindow *asplcan);
protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    AppWindow *splcan;
};
/*****
* Class AppWindow
*****/
class AppWindow : public IFrameWindow
{
public:
    AppWindow( unsigned long windowId);
    AppWindow();
    AppWindow & updateCAN(unsigned long eventtype);
private:
    ITitle * title;
    MySplit * splitCv;
    ACommandHandler * commandHandler;
};
#endif

```

RELATED CONCEPTS

“Split Canvases” on page 164
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Split Canvas” on page 165
 “File: splitcan.cpp” on page 170
 “File: splitcan.h” on page 173
 “File: splitcan.rc” on page 175
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

File: splitcan.rc

WIN

This is a Windows resource script file.

```
#include "splitcan.h"
/*****
/* Menu bar */
*****/
ID_MAIN MENUEX
BEGIN
    POPUP "&Actions" , ID_ACTIONS_MENU
    BEGIN
        MENUITEM "&Vertical orientation" , ID_VERT_ITEM
        MENUITEM "&Horizontal orientation" , ID_HORIZ_ITEM
        MENUITEM "", 0, MFT_SEPARATOR
        MENUITEM "&Double split bar edge" , ID_DOUBLE_EDGE_ITEM
        MENUITEM "H&alve split bar edge" , ID_HALVE_EDGE_ITEM
        MENUITEM "", 0, MFT_SEPARATOR
        MENUITEM "D&ouble split bar middle" , ID_DOUBLE_MIDDLE_ITEM
        MENUITEM "Ha&lve split bar middle" , ID_HALVE_MIDDLE_ITEM
    END
END
/*****
/* String table */
*****/
STRINGTABLE
BEGIN
    PSZ_OBJECT, "SPLIT"
    PSZ_VIEW, "Split Canvas"
    PSZ_TEXT1, "Static Text 1"
    PSZ_TEXT2, "Static Text 2"
    PSZ_TEXT3, "Static Text 3"
END
```

AIX

OS/2

This is an AIX or OS/2 resource script file:

```
#include "splitcan.h"
/*****
/* Menu bar */
*****/
MENU ID_MAIN
BEGIN
    SUBMENU "~Actions" , ID_ACTIONS_MENU
    BEGIN
        MENUITEM "~Vertical orientation" , ID_VERT_ITEM
        MENUITEM "~Horizontal orientation" , ID_HORIZ_ITEM
        MENUITEM SEPARATOR
        MENUITEM "~Double split bar edge" , ID_DOUBLE_EDGE_ITEM
        MENUITEM "H alve split bar edge" , ID_HALVE_EDGE_ITEM
        MENUITEM SEPARATOR
        MENUITEM "D ouble split bar middle" , ID_DOUBLE_MIDDLE_ITEM
        MENUITEM "Ha lve split bar middle" , ID_HALVE_MIDDLE_ITEM
    END
END
/*****
/* String table */
*****/
STRINGTABLE
BEGIN
    PSZ_OBJECT, "SPLIT"
    PSZ_VIEW, "Split Canvas"
```

```

    PSZ_TEXT1,    "Static Text 1"
    PSZ_TEXT2,    "Static Text 2"
    PSZ_TEXT3,    "Static Text 3"
END

```

RELATED CONCEPTS

"Split Canvases" on page 164
 "Chapter 21. Resources" on page 297
 "Chapter 1. Windows" on page 1
 "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a Split Canvas" on page 165
 "Convert Application Resources" on page 395
 "File: splitcan.cpp" on page 170
 "File: splitcan.h" on page 173
 "File: splitcan.hpp" on page 173
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

Split Canvas Example 2

File: esplitcv.cpp

```

//
// Split Canvas Example
// key functions:
//   - create a main window
//   - run the current application
//   - create horizontal and vertical split canvases
//   - create static text controls
//   - load strings from resource bound to the exe
//
#include <iapp.hpp>
#include "esplitcv.hpp"
#include "esplitcv.h"
/*****
* main
* - Application entry point.
*****/
int main()
{
    ASplitCanvas mainWindow(WND_MAIN);
    IApplication::current().run();
    return 0;
} /* end main */
/*****
* Class ASplitCanvas :: ASplitCanvas
*****/
ASplitCanvas :: ASplitCanvas( unsigned long windowId )
    : IFrameWindow( windowId )
    , horzCanvas( WND_CANVAS, this, this )
    , vertCanvas( WND_CANVAS2, &horzCanvas, &horzCanvas )
    , lText( WND_TXTL, &vertCanvas, &vertCanvas )
    , rText( WND_TXTR, &vertCanvas, &vertCanvas )
    , bText( WND_TXTB, &horzCanvas, &horzCanvas )
{
    // give the canvas a horizontal split bar
    // and make it the client area
    horzCanvas.setOrientation( ISplitCanvas::horizontalSplit );
    setClient( &horzCanvas );
    // give the canvas a vertical split bar
    vertCanvas.setOrientation( ISplitCanvas::verticalSplit );
}

```

```

// set top left static text
lText.setText(STR_TOPLEFT);
lText.setAlignment( IStaticText::centerCenter );
// set top right static text
rText.setText(STR_TOPRIGHT);
rText.setAlignment( IStaticText::centerCenter );
// set bottom static text
bText.setText(STR_BOTTOM);
bText.setAlignment( IStaticText::centerCenter );
// set focus and show window
setFocus().show();
} /* end ASplitCanvas :: ASplitCanvas(...) */

```

RELATED CONCEPTS

“Split Canvases” on page 164
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Split Canvas” on page 165
 “File: esplitcv.h”
 “File: esplitcv.hpp”
 “File: esplitcv.rc” on page 178
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

File: esplitcv.h

```

#ifndef ASPLITCV_H
#define ASPLITCV_H
#define WND_MAIN          0x1000
#define WND_CANVAS        0x1001
#define WND_CANVAS2       0x1002
#define WND_TXTB           0x1003
#define WND_TXTL           0x1004
#define WND_TXTR           0x1005
#define STR_BOTTOM         0x2000
#define STR_TOPLEFT       0x2001
#define STR_TOPRIGHT      0x2002
#endif

```

RELATED CONCEPTS

“Split Canvases” on page 164
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Split Canvas” on page 165
 “File: esplitcv.cpp” on page 176
 “File: esplitcv.hpp”
 “File: esplitcv.rc” on page 178
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

File: esplitcv.hpp

```

#ifndef ASPLITCV_HPP
#define ASPLITCV_HPP
#include <iframe.hpp>
#include <istattxt.hpp>
#include <isplitcv.hpp>

```

```

/*****
* Class ASplitCanvas                                     *
* - Time Handler class the processes time ticks for    *
*   AHelloWindow.                                       *
*****/
class ASplitCanvas : public IFrameWindow
{
public:
    ASplitCanvas(unsigned long windowId);
private:
    ISplitCanvas horzCanvas,
                vertCanvas;
    IStaticText lText,
                rText,
                bText;
};
#endif

```

RELATED CONCEPTS

"Split Canvases" on page 164
 "Chapter 1. Windows" on page 1
 "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a Split Canvas" on page 165
 "File: esplitcv.cpp" on page 176
 "File: esplitcv.h" on page 177
 "File: esplitcv.rc"
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

File: esplitcv.rc

```

#include "esplitcv.h"
/*****
* string resources                                     *
*   Relate symbolic names to literal text strings.    *
*****/
STRINGTABLE
BEGIN
    WND_MAIN      , "Canvas Classes Example 1 - Split Canvas"
    STR_BOTTOM    , "Bottom text"
    STR_TOPLEFT   , "Top left text"
    STR_TOPRIGHT  , "Top right text"
END

```

RELATED CONCEPTS

"Split Canvases" on page 164
 "Chapter 21. Resources" on page 297
 "Chapter 1. Windows" on page 1
 "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a Split Canvas" on page 165
 "Convert Application Resources" on page 395
 "File: esplitcv.cpp" on page 176
 "File: esplitcv.h" on page 177
 "File: esplitcv.hpp" on page 177
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

Multicell Canvases

The `IMultiCellCanvas` class lets you position child windows in a grid of cells based on the minimum sizes of the child windows.

Cells

A *cell* is the intersection of a row and a column. You may control the size of these cells, and how these cells behave when the user resizes a *multicell canvas*.

Expandable Rows and Columns

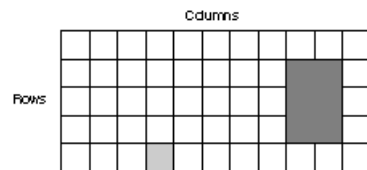
By default, the size of rows and columns is fixed. You may mark rows and columns as *expandable*. The multicell canvas resizes expandable rows and columns when the user resizes the canvas.

Positioning Child Windows

You place child windows on a multicell canvas by specifying the starting cell and the number of contiguous rows and columns that they span. You refer to cells in the grid by the column and row value. For example, you would refer to the top-left cell with the coordinate (1,1).

More than one child window may occupy the same cell, as long as they do not share the same *starting cell*. The starting cell is the start row and column you specify when you add a child window to a multicell canvas (by calling `addToCell`).

The following figure shows a multicell canvas. Notice that the small window's location is at column 4, row 5 (4,5), while the large window starts at column 9, row 2 (9,2). The large window is 2 columns wide and 3 rows high.



The default cell size is 10 pixels high by 10 pixels wide. The actual number of rows and columns in the canvas is the highest row and column value used. For example, suppose you placed a radio button placed at (4,5) and a push button at (2,7). Therefore, the canvas has 4 columns and 7 rows.

Sizing Child Windows

Generally, a multicell canvas does the following to size its child windows. These series of steps applies to rows, but analogous processing occurs for columns also:

1. The multicell canvas initially sizes rows to their minimum height (assigned by calls to `IMultiCellCanvas::setRowHeight`).
2. Any row without an explicit height gets the height of the default cell size. (This default is 10.)
3. The multicell canvas finds all child windows that occupy only a single row. For each such child window the multicell canvas does the following:
 - a. The canvas gets its minimum size.
 - b. If the height of the minimum size is larger than the height of the row the child window occupies, the multicell canvas sizes the row to the height of the minimum size.

After the multicell canvas checks all such child windows, the rows should have the greater of their assigned height (or default cell height), and the largest height of the child windows that they entirely contain.

4. The multicell canvas performs the following for all child windows that span multiple rows:
 - a. The canvas gets the child window's minimum size.
 - b. The canvas compares the height of the minimum size against the the sum of the heights of the rows the child window occupies.
 - c. If the sum is less, then do the following:
 - If none of the rows being spanned by the child window are expandable, the canvas increases the height of the first row to make the sum equal. (If the canvas was constructed with the `IMultiCellCanvas::spaceAddedToLast` style, it increases the height of the last row instead of the first.)
 - Otherwise, the canvas increases the height of the expandable row(s) to make the sum equal.
5. The multicell canvas increases the height of the expandable rows to fill the height of the canvas. (Expandable rows grow in proportion to their height.)
6. The multicell canvas sizes all child windows to fit their cells. However, you can cause the canvas to size the child window to its minimum size by assigning it an `IAlignmentAttribute` object. This also specify how the multicell canvas will align the child window within the cell.

Sizing the Multicell Canvas

The Open Class Library bases the minimum size of a multicell canvas by the minimum sizes of all of the windows plus the size of empty rows and columns. It does not include the extra space added to expandable rows and columns. If you size the multicell canvas smaller than this minimum size, you will clip the canvas at the lower-right corner. You can use the `IViewPort` class to add scroll bars to handle this situation. (If the multicell canvas has expandable rows or columns, you should construct the `IViewPort` with the `IViewPort::expandableViewWindow` style.)

Multicell Canvas Layout Aids

The multicell canvas has two styles that can help you create layouts with a multicell canvas. These styles provide a way to easily visualize the placement and size of each cell:

- *Grid lines*: Grid lines are stationary lines that the multicell canvas draws between rows and columns. Use the `gridLines` style to add grid lines to your multicell canvas.
- *Drag lines*: Drag lines are like grid lines, except that you may move them with the mouse. Use the `dragLines` style to add drag lines to your multicell canvas.

AIX

AIX does not support drag lines.

Here are some considerations when using grid lines and drag lines:

- The multicell canvas draws grid lines and drag lines at the left and top edges of cells.
- Child windows can overwrite grid lines and drag lines.
- The use of grid lines or drag lines does not change the initial placement or sizing of child windows.
- Moving a drag line resizes child windows on each side of the drag line in a manner similar to a split canvas.

RELATED CONCEPTS

- “Chapter 13. Canvas Controls” on page 155
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Create a Multicell Canvas”
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

Create a Multicell Canvas

The following example shows you how to create a multicell canvas. This canvas contains five static text fields and five entry fields:

1. Declare a class that uses a multicell canvas control in the header file with static text and entry field controls, as follows:

```
class MyWindow : public IFrameWindow
{
public:
    MyWindow(unsigned long windowID);
    MyWindow();
protected:
    bool command(ICommandEvent& cmdEvent);
private:
    IMultiCellCanvas myClient;
    IStaticText      stName,
                    stAddress,
                    stId,
                    stBirthday,
                    stPhone;
    IEntryField      efName,
                    efAddress,
                    efId,
                    efBirthday,
                    efPhone;
    ACommandHandler * commandHandler;
};
```

2. Construct the multicell canvas as follows:

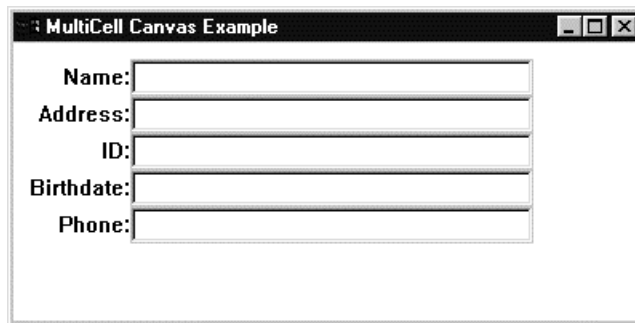
```
MyWindow::MyWindow( unsigned long windowId)
: IFrameWindow("MultiCell Canvas Example",windowId,
    IFrameWindow::defaultStyle()),
  myClient( ID_MCC, this, this),
  stName( ID_ST_NAME, &myClient, &myClient ),
  stAddress( ID_ST_ADDR, &myClient,& myClient ),
  stId( ID_ST_ID, &myClient, &myClient ),
  stBirthday( ID_ST_BD, &myClient, &myClient ),
  stPhone( ID_ST_PHONE, &myClient, &myClient ),
  efName( ID_EF_NAME, &myClient, &myClient ),
  efAddress( ID_EF_ADDR, &myClient, &myClient ),
  efId( ID_EF_ID, &myClient, &myClient ),
  efBirthday( ID_EF_BD, &myClient, &myClient ),
  efPhone( ID_EF_PHONE, &myClient, &myClient )
{
    setClient( &myClient );
    // Build multicell canvas with fields
    myClient.addToCell( &stName, 2, 2 );
    myClient.addToCell( &efName, 3, 2 );
    myClient.addToCell( &stAddress, 2, 3 );
    myClient.addToCell( &efAddress, 3, 3 );
    myClient.addToCell( &stId, 2, 4 );
    myClient.addToCell( &efId, 3, 4 );
```

```

myClient.addToCell( &stBirthday, 2, 5 );
myClient.addToCell( &efBirthday, 3, 5 );
myClient.addToCell( &stPhone, 2, 6 );
myClient.addToCell( &efPhone, 3, 6 );
// Load the static text fields
stName.setText( "Name:" );
stName.setForegroundColor(IColor::kBlack);
stName.setAlignment( IStaticText::centerRight );
stAddress.setText( "Address:" );
stAddress.setForegroundColor(IColor::kBlack);
stAddress.setAlignment( IStaticText::centerRight );
stBirthday.setText( "Birthdate:" );
stBirthday.setForegroundColor(IColor::kBlack);
stBirthday.setAlignment( IStaticText::centerRight );
stId.setText( "ID:" );
stId.setForegroundColor(IColor::kBlack);
stId.setAlignment( IStaticText::centerRight );
stPhone.setText( "Phone:" );
stPhone.setForegroundColor(IColor::kBlack);
stPhone.setAlignment( IStaticText::centerRight );
// Create Command Handler
commandHandler = new ACommandHandler(this);
commandHandler->handleEventsFor(this);
}

```

The following figure shows the multicell canvas that this example creates:



You can find the complete source to this example in the following files:

- mcc.cpp
- mcc.h
- mcc.hpp

The following example shows you how to create a window containing a multicell canvas. The canvas contains two check boxes, two radio buttons, three static text controls, and one push button:

1. This code from the header file declares the `AMultiCellCanvas` class as an `IFrameWindow` derived class:

```

class AMultiCellCanvas : public IFrameWindow
{
public:
    AMultiCellCanvas(unsigned long windowId);
    AMultiCellCanvas&
        displayButtonStatus();
private:
    IMultiCellCanvas    clientCanvas;
    IStaticText         status,
                        title1,
                        title2;
    ICheckBox           check1,

```

```

        check2;
IRadioButton    radio1,
                radio2;
IPushButton     pushButton;
APushButtonHandler  pushButtonHandler;
};

```

2. This code creates the window:

```

AMultiCellCanvas::AMultiCellCanvas(unsigned long windowId)
: IFrameWindow(windowId)
, clientCanvas( WND_MCCANVAS, this, this )
, status( WND_STATUS, &clientCanvas, &clientCanvas )
, title1( WND_TITLE1, &clientCanvas, &clientCanvas )
, title2( WND_TITLE2, &clientCanvas, &clientCanvas )
, check1( WND_CHECK1, &clientCanvas, &clientCanvas )
, check2( WND_CHECK2, &clientCanvas, &clientCanvas )
, radio1( WND_RADIO1, &clientCanvas, &clientCanvas )
, radio2( WND_RADIO2, &clientCanvas, &clientCanvas )
, pushButton( WND_PUSHBUT, &clientCanvas, &clientCanvas )
{
    // make multi-cell canvas the client
    setClient( &clientCanvas );
    // set status area text
    status.setAlignment( IStaticText::centerCenter );
    status.setText( STR_STATUS );
    // set text and attributes
    title1.setAlignment( IStaticText::centerLeft );
    title1.setText( STR_TITLE1 );
    // set text and attributes
    title2.setAlignment( IStaticText::centerLeft );
    title2.setText( STR_TITLE2 );
    // set checkbox text
    check1.setText( STR_CHECK1 );
    // set radio button text
    check2.setText( STR_CHECK2 );
    radio1.setText( STR_RADIO1 );
    radio2.setText( STR_RADIO2 );
    pushButton.setText( STR_PUSHBUT );
    // pre-select one radio button
    radio1.select();
    // set tabStop and Group styles
    check1.enableGroup().enableTabStop();
    radio1.enableGroup().enableTabStop();
    pushButton.enableGroup().enableTabStop();
    // initialize push button handler
    pushButtonHandler.setOwnerWindow(this);
    // add handler to canvas
    pushButtonHandler.handleEventsFor(&clientCanvas);
    // add controls to canvas.
    // the canvas runs from 1,1 to 4,7
    // exactly one row and one column is expandable, as this
    // allows the canvas to fill the whole client.
    clientCanvas.addToCell(&status , 1, 1, 4, 1);
    clientCanvas.addToCell(&title1 , 1, 3, 2, 1);
    clientCanvas.addToCell(&title2 , 3, 3, 2, 1);
    clientCanvas.addToCell(&check1 , 2, 4);
    clientCanvas.addToCell(&check2 , 2, 5);
    clientCanvas.addToCell(&radio1 , 4, 4);
    clientCanvas.addToCell(&radio2 , 4, 5);
    clientCanvas.addToCell(&pushButton , 2, 7);
    // set size of empty rows
    clientCanvas.setRowHeight(2, 20, true);
    clientCanvas.setRowHeight(6, 40);
    // last column expandable
    clientCanvas.setColumnWidth(4, 40, true);
    // set focus to first checkbox

```

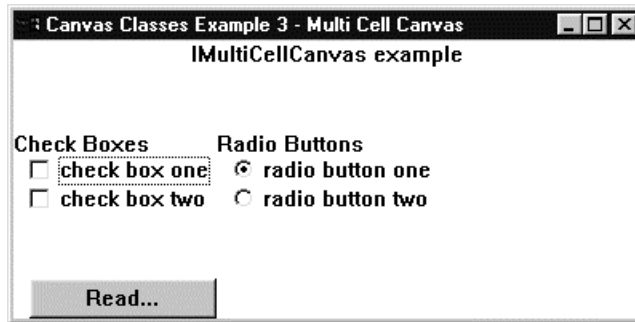
```

    check1.setFocus();
    // show main window
    show();
}

```

Note that there is not a 1:1 relationship between Windows, OS/2 and Motif pixels.

The following figure shows the completed multicell canvas:



You can find the complete source to this example in the following files:

- emcelcv.cpp
- emcelcv.h
- emcelcv.hpp
- emcelcv.rc

RELATED CONCEPTS

- “Chapter 13. Canvas Controls” on page 155
- “Multicell Canvases” on page 179

RELATED TASKS

- “Create a Set Canvas” on page 157
- “Create a Split Canvas” on page 165
- “Create a View Port” on page 193

RELATED REFERENCES

- “File: mcc.cpp”
- “File: mcc.h” on page 186
- “File: mcc.hpp” on page 187
- “File: emcelcv.cpp” on page 188
- “File: emcelcv.h” on page 190
- “File: emcelcv.hpp” on page 191
- “File: emcelcv.rc” on page 192

Multicell Canvas Example 1

File: mcc.cpp

```

#include "mcc.hpp"
#include "mcc.h"
#include <iostream.h>
/*****
* main
*

```

```

* - Application entry point.
*****/
int main()
{
    MyWindow myWindow(WND_MAIN);
    myWindow.show();
    IApplication::current().run();
    return 0;
}
/*****
* Class MyWindow :: MyWindow
*****/
MyWindow::MyWindow( unsigned long windowId)
: IFrameWindow("MultiCell Canvas Example",windowId,
               IFrameWindow::defaultStyle()),
  myClient( ID_MCC, this, this),
  stName( ID_ST_NAME, &myClient, &myClient ),
  stAddress( ID_ST_ADDR, &myClient, &myClient ),
  stId( ID_ST_ID, &myClient, &myClient ),
  stBirthday( ID_ST_BD, &myClient, &myClient ),
  stPhone( ID_ST_PHONE, &myClient, &myClient ),
  efName( ID_EF_NAME, &myClient, &myClient ),
  efAddress( ID_EF_ADDR, &myClient, &myClient ),
  efId( ID_EF_ID, &myClient, &myClient ),
  efBirthday( ID_EF_BD, &myClient, &myClient ),
  efPhone( ID_EF_PHONE, &myClient, &myClient )
{
    setClient( &myClient );
    // Build multicell canvas with fields
    myClient.addToCell( &stName, 2, 2 );
    myClient.addToCell( &efName, 3, 2 );
    myClient.addToCell( &stAddress, 2, 3 );
    myClient.addToCell( &efAddress, 3, 3 );
    myClient.addToCell( &stId, 2, 4 );
    myClient.addToCell( &efId, 3, 4 );
    myClient.addToCell( &stBirthday, 2, 5 );
    myClient.addToCell( &efBirthday, 3, 5 );
    myClient.addToCell( &stPhone, 2, 6 );
    myClient.addToCell( &efPhone, 3, 6 );
    // Load the static text fields
    stName.setText( "Name:" );
    stName.setForegroundColor(IColor::kYellow);
    stName.setAlignment( IStaticText::centerRight );
    stAddress.setText( "Address:" );
    stAddress.setForegroundColor(IColor::kYellow);
    stAddress.setAlignment( IStaticText::centerRight );
    stBirthday.setText( "Birthdate:" );
    stBirthday.setForegroundColor(IColor::kYellow);
    stBirthday.setAlignment( IStaticText::centerRight );
    stId.setText( "ID:" );
    stId.setForegroundColor(IColor::kYellow);
    stId.setAlignment( IStaticText::centerRight );
    stPhone.setText( "Phone:" );
    stPhone.setForegroundColor(IColor::kYellow);
    stPhone.setAlignment( IStaticText::centerRight );
    // Create Command Handler
    commandHandler = new ACommandHandler(this);
    commandHandler->handleEventsFor(this);
}
/*****
* Class MyWindow :: ~MyWindow
*****/
MyWindow::~MyWindow ()
{
    commandHandler->stopHandlingEventsFor(this);
}
/*****

```

```

* Class ACommandHandler :: ACommandHandler *
*****/
ACommandHandler::ACommandHandler(MyWindow *amcc)
{
    mcc_type = amcc;
}
/*****
* Class ACommandHandler :: command *
*****/
bool ACommandHandler::command(ICommandEvent& cmdEvent)
{
    switch (cmdEvent.commandId())
    {
        case ID_DUMMY:
            return true;
            break;
        default:
            return false;
    }
    return 0;
}

```

RELATED CONCEPTS

“Multicell Canvases” on page 179
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Multicell Canvas” on page 181
 “File: mcc.h”
 “File: mcc.hpp” on page 187
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

File: mcc.h

```

#ifndef MCELLH_
#define MCELLH_
#define WND_MAIN 1
#define ID_DUMMY 2
#define ID_MCC 3
#define ID_ST_NAME 4
#define ID_ST_ADDR 5
#define ID_ST_ID 6
#define ID_ST_BD 7
#define ID_ST_PHONE 8
#define ID_EF_NAME 9
#define ID_EF_ADDR 10
#define ID_EF_ID 11
#define ID_EF_BD 12
#define ID_EF_PHONE 13
#endif

```

RELATED CONCEPTS

“Multicell Canvases” on page 179
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Multicell Canvas” on page 181
 “File: mcc.cpp” on page 184
 “File: mcc.hpp”
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

File: mcc.hpp

```

#ifndef _MCELL_
#define _MCELL_
#include <iframe.hpp>
#include <icmdhdr.hpp>
#include <iapp.hpp>
#include <imcelcv.hpp>
#include <imle.hpp>
#include <iinfoa.hpp>
#include <istattxt.hpp>
#include <ientryfd.hpp>
class ACommandHandler;
class IFrameWindow;
class MyWindow;
/*****
 * Class ACommandHandler
 *****/
class ACommandHandler : public ICommandHandler
{
public:
    ACommandHandler(MyWindow *amcc);
protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    MyWindow *mcc_type;
};
/*****
 * Class MyWindow
 *****/
class MyWindow : public IFrameWindow
{
public:
    MyWindow(unsigned long windowID);
    MyWindow();
protected:
    bool command(ICommandEvent& cmdEvent);
private:
    IMultiCellCanvas myClient;
    IStaticText      stName,
                    stAddress,
                    stId,
                    stBirthday,
                    stPhone;
    IEntryField      efName,
                    efAddress,
                    efId,
                    efBirthday,
                    efPhone;
    ACommandHandler * commandHandler;
};
#endif

```

RELATED CONCEPTS

“Multicell Canvases” on page 179
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Multicell Canvas” on page 181

“File: mcc.cpp” on page 184

“File: mcc.h” on page 186

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Multicell Canvas Example 2

File: emcelcv.cpp

```
//
// Multi Cell Canvas
// key functions:
// - create a main window
// - run the current application
// - create a multi cell canvas and use as client area
//     canvas has 4 columns and 7 rows
//     one row and one column are expandable
//     all other controls attached to canvas
// - create static text controls
// - create check boxes, radio buttons and push buttons
// - control cursor and tab movement between different groups
// - process 'Command' events generated by the push button
// - load strings from resource bound to the exe
//
#include <iapp.hpp>
#include <imgbox.hpp>
#include <istring.hpp>
#include <ireslib.hpp>
#include <ifont.hpp>
#include "emcelcv.h"
#include "emcelcv.hpp"
/*****
* main
* - Application entry point.
*****/
int main()
{
    AMultiCellCanvas mainWindow(WND_MAIN);
    IApplication::current().run();
    return 0;
}
/*****
* Class AMultiCellCanvas :: AMultiCellCanvas
*****/
AMultiCellCanvas::AMultiCellCanvas(unsigned long windowId)
: IFrameWindow(windowId)
, clientCanvas( WND_MCCANVAS, this, this )
, status( WND_STATUS, &clientCanvas, &clientCanvas )
, title1( WND_TITLE1, &clientCanvas, &clientCanvas )
, title2( WND_TITLE2, &clientCanvas, &clientCanvas )
, check1( WND_CHECK1, &clientCanvas, &clientCanvas )
, check2( WND_CHECK2, &clientCanvas, &clientCanvas )
, radiol( WND_RADIO1, &clientCanvas, &clientCanvas )
, radio2( WND_RADIO2, &clientCanvas, &clientCanvas )
, pushButton( WND_PUSHBUT, &clientCanvas, &clientCanvas )
{
    // make multi-cell canvas the client
    setClient( &clientCanvas );
    // set status area text
    status.setAlignment( IStaticText::centerCenter );
    status.setText( STR_STATUS );
    // set text and attributes
```



```

title1.setAlignment( IStaticText::centerLeft );
title1.setText( STR_TITLE1 );
// set text and attributes
title2.setAlignment( IStaticText::centerLeft );
title2.setText( STR_TITLE2 );
// set checkbox text
check1.setText( STR_CHECK1 );
// set radio button text
check2.setText( STR_CHECK2 );
radio1.setText( STR_RADIO1 );
radio2.setText( STR_RADIO2 );
pushButton.setText( STR_PUSHBUT );
// pre-select one radio button
radio1.select();
// set tabStop and Group styles
check1.enableGroup().enableTabStop();
radio1.enableGroup().enableTabStop();
pushButton.enableGroup().enableTabStop();
// initialize push button handler
pushButtonHandler.setOwnerWindow(this);
// add handler to canvas
pushButtonHandler.handleEventsFor(&clientCanvas);
// add controls to canvas.
// the canvas runs from 1,1 to 4,7
// exactly one row and one column is expandable, as this
// allows the canvas to fill the whole client.
clientCanvas.addToCell(&status , 1, 1, 4, 1);
clientCanvas.addToCell(&title1 , 1, 3, 2, 1);
clientCanvas.addToCell(&title2 , 3, 3, 2, 1);
clientCanvas.addToCell(&check1 , 2, 4);
clientCanvas.addToCell(&check2 , 2, 5);
clientCanvas.addToCell(&radio1 , 4, 4);
clientCanvas.addToCell(&radio2 , 4, 5);
clientCanvas.addToCell(&pushButton , 2, 7);
// set size of empty rows
clientCanvas.setRowHeight(2, 20, true);
clientCanvas.setRowHeight(6, 40);
// last column expandable
clientCanvas.setColumnWidth(4, 40, true);
// set focus to first checkbox
check1.setFocus();
// show main window
show();
} /* end AMultiCellCanvas :: AMultiCellCanvas(...) */
/*****
* Class AMultiCellCanvas :: displayButtonStatus
*****/
AMultiCellCanvas& AMultiCellCanvas::displayButtonStatus()
{
    unsigned long str_id1, str_id2, str_id3;
    IMessageBox msgbox(this);
    IResourceLibrary reslib =
        IApplication::current().userResourceLibrary();
    msgbox.setTitle( IResourceId(STR_MSGBOX) );
    str_id1 = check1.isSelected() ? STR_CHK1_SEL : STR_CHK1_NOSEL;
    str_id2 = check2.isSelected() ? STR_CHK2_SEL : STR_CHK2_NOSEL;
    IString str1 = reslib.loadString(str_id1),
            str2 = reslib.loadString(str_id2),
            str3 = reslib.loadString(radio1.selectedIndex() +
                                   STR_RAD1_SEL );
    str1 += str2 + str3;
    msgbox.show( (char *)str1 , IMessageBox::okButton
                IMessageBox::informationIcon
                IMessageBox::applicationModal
                IMessageBox::moveable );
    return *this;
}

```

```

/*****
* Class APushButtonHandler :: command
*****/
bool APushButtonHandler::command( ICommandEvent& evt )
{
    bool fProcessed = false;
    if (evt.commandId() == WND_PUSHBUT && window)
    {
        window->displayButtonStatus();
        fProcessed = true;
    }
    return fProcessed;
}

```

RELATED CONCEPTS

"Multicell Canvases" on page 179
 "Chapter 1. Windows" on page 1
 "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a Multicell Canvas" on page 181
 "File: emcelcv.h"
 "File: emcelcv.hpp" on page 191
 "File: emcelcv.rc" on page 192
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

File: emcelcv.h

```

#ifndef AMCELCV_H
#define AMCELCV_H
#define WND_MAIN          0x1000
#define WND_STATUS       0x1001
#define WND_MCCANVAS     0x1002
#define WND_TITLE1       0x1003
#define WND_TITLE2       0x1004
#define WND_RADIO1       0x1005
#define WND_RADIO2       0x1006
#define WND_CHECK1       0x1007
#define WND_CHECK2       0x1008
#define WND_PUSHBUT     0x1009
#define STR_STATUS       0x2000
#define STR_TITLE1       0x2001
#define STR_TITLE2       0x2002
#define STR_CHECK1       0x2003
#define STR_CHECK2       0x2004
#define STR_RADIO1       0x2005
#define STR_RADIO2       0x2006
#define STR_PUSHBUT     0x2007
#define STR_PUSHED       0x2008
#define STR_MSGBOX       0x2009
#define STR_CHK1_SEL     0x200A
#define STR_CHK1_NOSEL   0x200B
#define STR_CHK2_SEL     0x200C
#define STR_CHK2_NOSEL   0x200D
#define STR_RAD1_SEL     0x200E
#define STR_RAD2_SEL     0x200F
#endif

```

RELATED CONCEPTS

"Multicell Canvases" on page 179
 "Chapter 1. Windows" on page 1
 "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a Multicell Canvas" on page 181
 "File: emcelcv.cpp" on page 188
 "File: emcelcv.hpp"
 "File: emcelcv.rc" on page 192
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

File: emcelcv.hpp

```

#ifndef AMCELCV_HPP
#define AMCELCV_HPP
#include <iframe.hpp>
#include <istattxt.hpp>
#include <ipushbut.hpp>
#include <iradiobt.hpp>
#include <icheckbx.hpp>
#include <imcelcv.hpp>
#include <icmdhdr.hpp>
// forward declaration of class
class AMultiCellCanvas;
/*****
 * Class APushButtonHandler
 * - The push button handler uses the displayButtonStatus
 *   function from the multicell canvas when the button is
 *   pressed.
 *****/
class APushButtonHandler : public ICommandHandler
{
public:
    APushButtonHandler() : window(0) {};
    setOwnerWindow(AMultiCellCanvas *pmcellcv)
        { window = pmcellcv; return 0; }
protected:
    command( ICommandEvent& evt );
private:
    AMultiCellCanvas * window;
};
/*****
 * Class AMultiCellCanvas
 * - Main window for the application. It contains a
 *   multicell canvas and the other controls are attached
 *   to the canvas. The pushbutton displays a message box
 *   showing the current state of the buttons
 *****/
class AMultiCellCanvas : public IFrameWindow
{
public:
    AMultiCellCanvas(unsigned long windowId);
    AMultiCellCanvas&
        displayButtonStatus();
private:
    IMultiCellCanvas    clientCanvas;
    IStaticText         status,
                       title1,
                       title2;
    ICheckBox           check1,
                       check2;
    IRadioButton        radiol1,
                       radio2;

```

```

        IPushButton      pushButton;
        APushButtonHandler  pushButtonHandler;
};
#endif

```

RELATED CONCEPTS

"Multicell Canvases" on page 179
 "Chapter 1. Windows" on page 1
 "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a Multicell Canvas" on page 181
 "File: emcelcv.cpp" on page 188
 "File: emcelcv.h" on page 190
 "File: emcelcv.rc"
 "Write an Event Handler" on page 25
 "Task and Samples Cross-Reference Table" on page 445

File: emcelcv.rc

```

#include "emcelcv.h"
/*****
 * string resources
 * Relate symbolic names to literal text strings.
 *****/
STRINGTABLE
BEGIN
    WND_MAIN          , "Canvas Classes Example 3 - Multi Cell Canvas"
    STR_STATUS        , "IMultiCellCanvas example"
    STR_TITLE1        , "Check Boxes"
    STR_TITLE2        , "Radio Buttons"
    STR_CHECK1        , "check box one"
    STR_CHECK2        , "check box two"
    STR_RADIO1        , "radio button one"
    STR_RADIO2        , "radio button two"
    STR_PUSHBUT       , "Read..."
    STR_PUSHED        , "You have pressed the push button!"
    STR_MSGBOX        , "Current Button State"
    STR_CHK1_SEL      , "check box one selected, "
    STR_CHK1_NOSEL    , "check box one not selected, "
    STR_CHK2_SEL      , "check box two selected, "
    STR_CHK2_NOSEL    , "check box two not selected, "
    STR_RAD1_SEL      , "and radio button one selected"
    STR_RAD2_SEL      , "and radio button two selected"
END

```

RELATED CONCEPTS

"Multicell Canvases" on page 179
 "Chapter 21. Resources" on page 297
 "Chapter 1. Windows" on page 1
 "Chapter 3. Events and Event Handlers" on page 19

RELATED TASKS

"Create a Multicell Canvas" on page 181
 "Convert Application Resources" on page 395
 "File: emcelcv.cpp" on page 188
 "File: emcelcv.h" on page 190
 "File: emcelcv.hpp" on page 191

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

View Ports

The `IViewPort` class provides a scrollable view area with horizontal and vertical scroll bars. A user may scroll a view port using the keyboard, or the mouse if the view port displays any scroll bars.

A view port consists of four windows:

- The horizontal scroll bar.
- The vertical scroll bar.
- The *clipping rectangle*. This is the rectangular area that the scroll bars do not occupy, or the area within the view port where you view your data.
- The view port. This is the parent and the owner of the windows listed above.

The *view window* is the child window that a view port scrolls.

Behavior and Characteristics

By default, a *view port* will display scroll bars only when needed: if the view window is larger than the clipping window, the user can scroll the view window. You may specify whether the view port must always display either of its scroll bars (by using the styles `IViewPort::alwaysHorizontalScrollBar` or `IViewPort::alwaysVerticalScrollBar`). You may also specify whether the view port must never display either of its scroll bars (by using the styles `IViewPort::noHorizontalScrollBar` or `IViewPort::noVerticalScrollBar`).

The size of the child window is fixed (unless you construct the viewport with the `IViewPort::expandableViewWindowStyle`).

A view port can have only one child window (unless you call `IViewPort::setViewWindow`). To display more than one window in a view port at the same time, place the windows into another type of canvas. Make this canvas the child of the view port.

RELATED CONCEPTS

“Chapter 13. Canvas Controls” on page 155

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a View Port”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Create a View Port

A *view port* canvas provides a scrollable view area with horizontal and vertical scroll bars. By default, the scroll bars display only when needed. A view port can have only one child control. The size of the child control is fixed. If the view port is smaller than the child control, the view port allows the user to scroll the child control.

If you need more than one control in a view port, place the controls into another type of canvas, which you can then make the child of the view port.

The following example shows how to create a view port.

1. Declare a view port as follows:

```
class AViewWindow : public IFrameWindow
{
public:
    AViewWindow(unsigned long windowId);
    ~AViewWindow();
    AViewWindow & moveHoriz();
    AViewWindow & moveVert();
private:
    ITitle        title;
    IViewPort     viewPort;
    ICanvas       canvas;
    IStaticText   text;
    IMultiLineEdit mle1;
    IMultiLineEdit mle2;
    ACommandHandler * commandHandler;
};
```

2. Define the view port as shown in the following code:

```
AViewWindow::AViewWindow(unsigned long windowId)
: IFrameWindow(windowId,
  IFrameWindow::defaultStyle() |
  IFrameWindow::menuBar),
  title(this, "View Port Example"),
  viewPort(WID_VIEWPORT, this, this, IRectangle()),
  canvas(WID_CANVAS, &viewPort, &viewPort,
    IRectangle(5,15, 410, 660)),
  text(WID_TEXT, &canvas, &canvas,
    IRectangle(10, 360, 400, 410)),
  mle1(WID_MLE1, &canvas, &canvas,
    IRectangle(10,10, 400, 149)),
  mle2(WID_MLE2, &canvas, &canvas,
    IRectangle(10, 160, 400, 349))
{
    viewPort.setBackgroundColor(IColor(199, 212, 204));
    setClient(&viewPort);
    text.setText("Enter some data into the MLE, please");
    commandHandler = new ACommandHandler(this);
    commandHandler->handleEventsFor(this);
    sizeTo(ISize(400,500));
    show();
}
```

3. Handle events, as follows:

```

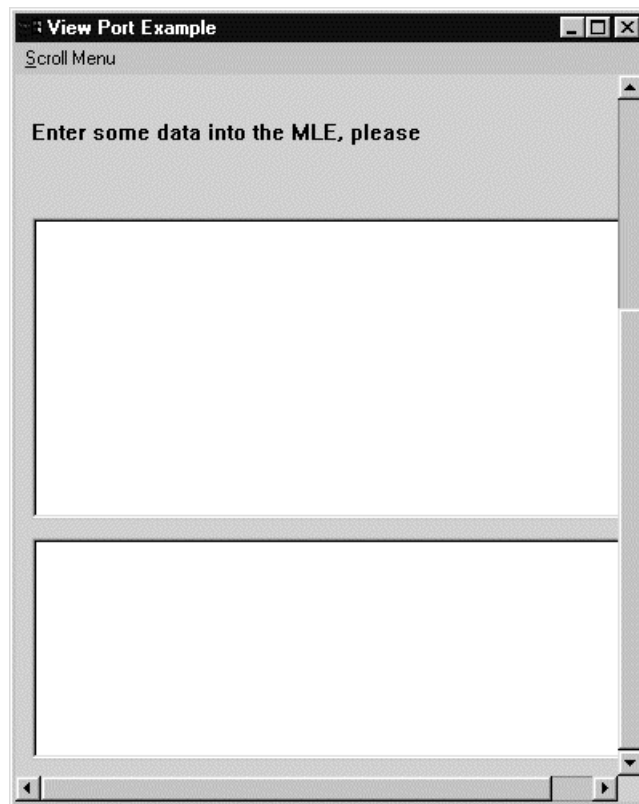
/*****
 * Class AViewWindow :: moveHoriz
 *****/
AViewWindow & AViewWindow :: moveHoriz()
{
    viewPort.scrollViewHorizontallyTo(200);
    return(*this);
}
/*****
 * Class AViewWindow :: moveVert
 *****/
AViewWindow & AViewWindow :: moveVert()
{
    viewPort.scrollViewVerticallyTo(200);
    return(*this);
}
/*****
 * Class ACommandHandler :: ACommandHandler
 *****/
ACommandHandler::ACommandHandler(AViewWindow *viewWindow)
{
```

```

    view = viewWindow;
}
/*****
* Class ACommandHandler :: command
*****/
bool ACommandHandler::command(ICommandEvent & cmdEvent)
{
    bool eventProcessed(true);
    // Identify the event
    switch (cmdEvent.commandId())
    {
        case ID_HORIZ:
            view->moveHoriz();
            break;
        case ID_VERT:
            view->moveVert();
            break;
        default:
            eventProcessed = false;
    }
    return true;
}

```

The following figure shows the view port created with the preceding code.



You can find the source to this example in the following files:

- vp1.cpp
- vp1.h
- vp1.hpp
- vp1.rc

RELATED CONCEPTS

“View Ports” on page 193
“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a Set Canvas” on page 157
“Create a Split Canvas” on page 165
“Create a Multicell Canvas” on page 181
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

“File: vp1.cpp”
“File: vp1.h” on page 198
“File: vp1.hpp” on page 198
“File: vp1.rc” on page 199

View Port Example

File: vp1.cpp

```
#include "vp1.h"
#include "vp1.hpp"
#include <icoordsy.hpp>
/*****
 * main
 * - Application entry point.
 *****/
int main()
{
    ICoordinateSystem::setApplicationOrientation(
        ICoordinateSystem::kOriginLowerLeft);
    AViewWindow viewWindow(WID_MAIN);
    IApplication::current().run();
    return 0;
}
/*****
 * Class AViewWindow :: AViewWindow
 *****/
AViewWindow::AViewWindow(unsigned long windowId)
: IFrameWindow(windowId,
    IFrameWindow::defaultStyle() |
    IFrameWindow::menuBar),
  title(this, "View Port Example"),
  viewPort(WID_VIEWPORT, this, this, IRectangle()),
  canvas(WID_CANVAS, &viewPort, &viewPort,
    IRectangle(5,15, 410, 660)),
  text(WID_TEXT, &canvas, &canvas,
    IRectangle(10, 360, 400, 410)),
  m1e1(WID_MLE1, &canvas, &canvas,
    IRectangle(10,10, 400, 149)),
  m1e2(WID_MLE2, &canvas, &canvas,
    IRectangle(10, 160, 400, 349))
{
    viewPort.setBackgroundColor(IColor(199, 212, 204));
    setClient(&viewPort);
    text.setText("Enter some data into the MLE, please");
    commandHandler = new ACommandHandler(this);
    commandHandler->handleEventsFor(this);
    sizeTo(ISize(400,500));
    show();
}
```



```

}
/*****
* Class AViewWindow :: ~AViewWindow
*****/
AViewWindow::~AViewWindow()
{
    commandHandler->stopHandlingEventsFor(this);
    delete commandHandler;
}
/*****
* Class AViewWindow :: moveHoriz
*****/
AViewWindow & AViewWindow :: moveHoriz()
{
    viewport.scrollViewHorizontallyTo(200);
    return(*this);
}
/*****
* Class AViewWindow :: moveVert
*****/
AViewWindow & AViewWindow :: moveVert()
{
    viewport.scrollViewVerticallyTo(200);
    return(*this);
}
/*****
* Class ACommandHandler :: ACommandHandler
*****/
ACommandHandler::ACommandHandler(AViewWindow *viewWindow)
{
    view = viewWindow;
}
/*****
* Class ACommandHandler :: command
*****/
bool ACommandHandler::command(ICommandEvent & cmdEvent)
{
    bool eventProcessed(true);
    // Identify the event
    switch (cmdEvent.commandId())
    {
        case ID_HORIZ:
            view->moveHoriz();
            break;
        case ID_VERT:
            view->moveVert();
            break;
        default:
            eventProcessed = false;
    }
    return true;
}

```

RELATED CONCEPTS

“View Ports” on page 193

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a View Port” on page 193

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

“File: radio.h” on page 110
“File: radio.hpp” on page 110
“File: radio.rc” on page 111

File: vp1.h

```
#ifndef _VIEWPORT_H_
#define _VIEWPORT_H_
/*****
 * Window IDs
 *****/
#define WID_MAIN          0x1000
#define WID_VIEWPORT     0x1010
#define WID_CANVAS       0x1020
#define WID_TEXT         0x1030
#define WID_MLE1         0x1040
#define WID_MLE2         0x1050
/*****
 * Menu IDs
 *****/
#define ID_MENU           0x2000
#define ID_HORIZ         0x2002
#define ID_VERT          0x2003
#endif
```

RELATED CONCEPTS

“View Ports” on page 193
“Chapter 1. Windows” on page 1
“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a View Port” on page 193
“Write an Event Handler” on page 25
“Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

“File: radio.cpp” on page 107
“File: radio.hpp” on page 110
“File: radio.rc” on page 111

File: vp1.hpp

```
#ifndef _VIEWPORT_
#define _VIEWPORT_
#include <iframe.hpp>
#include <icmdhdr.hpp>
#include <iapp.hpp>
#include <ititle.hpp>
#include <imle.hpp>
#include <istattxt.hpp>
#include <icanvas.hpp>
#include <ivport.hpp>
// Forward declarations:
class IFrameWindow;
class AViewWindow;
class ICommandHandler;
/*****
 * Class ACommandHandler
 *****/
```

```

class ACommandHandler : public ICommandHandler
{
public:
    ACommandHandler(AViewWindow *viewWindow);
protected:
    virtual bool command(ICommandEvent& cmdEvent);
private:
    AViewWindow *view;
};
/*****
* Class AViewWindow
*****/
class AViewWindow : public IFrameWindow
{
public:
    AViewWindow(unsigned long windowId);
    AViewWindow();
    AViewWindow & moveHoriz();
    AViewWindow & moveVert();
private:
    ITitle      title;
    IViewPort   viewPort;
    ICanvas     canvas;
    IStaticText text;
    IMultiLineEdit mle1;
    IMultiLineEdit mle2;
    ACommandHandler * commandHandler;
};
#endif

```

RELATED CONCEPTS

“View Ports” on page 193
 “Chapter 1. Windows” on page 1
 “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a View Port” on page 193
 “Write an Event Handler” on page 25
 “Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

“File: radio.cpp” on page 107
 “File: radio.h” on page 110
 “File: radio.rc” on page 111

File: vp1.rc

WIN

This is a Windows resource script file:

```

#include "vp1.h"
WID_MAIN MENUEX
BEGIN
    POPUP "&Scroll Menu"          , ID_MENU
    BEGIN
        MENUITEM "&HorizontalShift" , ID_HORIZ
        MENUITEM "&VerticalShift"   , ID_VERT
    END
END

```

> AIX

> OS/2

This is an AIX or OS/2 resource script file:

```
#include "vp1.h"
MENU WID_MAIN
  BEGIN
    SUBMENU "~Scroll Menu"          , ID_MENU
      BEGIN
        MENUITEM "~HorizontalShift" , ID_HORIZ
        MENUITEM "~VerticalShift"   , ID_VERT
      END
    END
  END
```

RELATED CONCEPTS

“View Ports” on page 193

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Create a View Port” on page 193

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

RELATED REFERENCES

“File: radio.cpp” on page 107

“File: radio.h” on page 110

“File: radio.hpp” on page 110

Drawing Canvases

The `IDrawingCanvas` class is a control class that provides support for drawing graphic objects.

Behavior and Characteristics

The *drawing canvas* contains an `IGraphicGroup` to which you can add graphic objects. By default, Open Class Library attaches a paint handler 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 graphic objects in the `IGraphicGroup`, redrawing the graphic objects necessary to update the window. If you do not add graphic objects to the `IGraphicGroup`, the drawing canvas simply paints the window using the current background color.

To add graphic objects to a drawing canvas, create an `IGraphicGroup` object and call `IViewPort::setGraphicGroup`. You can easily change the objects drawn in the drawing canvas by changing the `IGraphicGroup` object.

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.

If the user resizes the drawing canvas smaller, it does not repaint. If your application requires the drawing canvas to repaint in this case, use the `IResizeHandler` class to call the `refresh` member function.

Like the ICanvas class, the IDrawingCanvas class does not position or size child windows. All drawing will be done underneath any child windows.

Graphic Contexts

You have the option of setting the *graphic context*, or the drawing port, that the drawing canvas will use when drawing the graphic objects contained in the IGraphicGroup object. By changing graphic contexts, you can change the appearance of the graphic objects or the device on which they are rendered. (You would use the setGrafPort member function to do this.)

RELATED CONCEPTS

2D Graphics Overview

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

Chapter 14. Notebooks

A *notebook* control is a visual component that organizes related information on individual “pages” so that a user can find and display that information. It simulates a real notebook and provides the user with a recognizable visual component. Users select and display pages using a mouse or the keyboard. Your application can specify different colors, sizes, and orientations for its notebooks, but the underlying function of the control remains the same.

WIN




The INotebook class supports both a Presentation Manager-compatible notebook control and a Windows tab control. The PM-compatible version allows code using INotebook to be shared easily between OS/2 and Windows. However, the look and feel of the notebook control on Windows will not match the look and feel of other applications.

AIX

Use INotebook if you might port your code across different platforms since the ITabControl class is not available on AIX.

A notebook generally contains the following components:

Component	Description
Page window area	The notebook reserves the <i>client area</i> to display the application window.
Binding	You may choose to display your notebook with either a spiral <i>binding</i> or a solid binding. This ornamental feature visually depicts the spine of your notebook. WIN The Windows tab control contains no binding area. Windows ignores any binding settings.
Back pages	The <i>back pages</i> are the recessed edges that give the notebook a three-dimensional look. WIN The Windows tab control contains no back pages area. Windows ignores any back pages settings.
Page buttons	The <i>page buttons</i> allow the user to move through the pages of the notebook one page at a time. The notebook positions the page buttons on the corner where the back pages intersect.

Component	Description
Major tabs	<p>The <i>major tabs</i> let the user move quickly through the sections of the notebook. You may orient the major tabs along any edge of the notebook. The INotebook class will put the binding on the opposite edge. You may also change the shape of the tabs.</p>  <p>The Windows tab control ignores shape settings and tab orientation. It only supports rounded tabs positioned at the top of the control. The Windows tab control supports multiple rows of tabs; this style keeps all tabs visible to the user.</p>
Minor tabs	<p>The <i>minor tabs</i> permit the user to move within a major tab section of a notebook. The notebook places the minor tabs along the back page edge that does not already hold the major tabs.</p>  <p>The Windows tab control does not support minor tabs. As a result, the INotebook class converts all pages in a notebook with minor tabs or no tabs to pages with major tabs.</p>
Status area	<p>The <i>status area</i> displays a text string about the current page. The notebook positions the status area on the same line as the page buttons.</p>  <p>The Windows tab control contains no status area. Windows ignores any status area settings.</p>

RELATED CONCEPTS

“Tab Controls” on page 214

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

“Use Notebooks”

“Hello World Version 6: Adding a Font Dialog, Pop-up Menu, Notebooks, and Graphics” on page 443

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

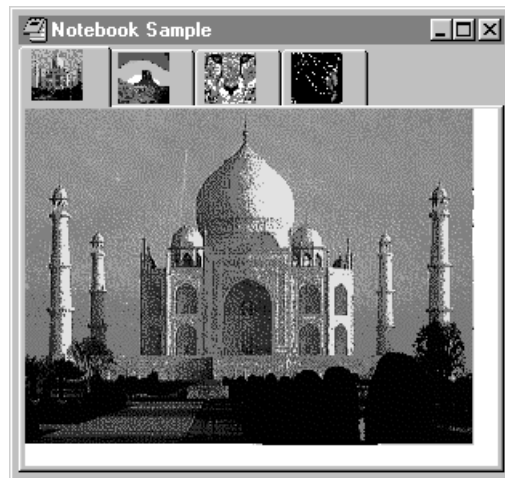
Use Notebooks

Using the Default Notebook Styles

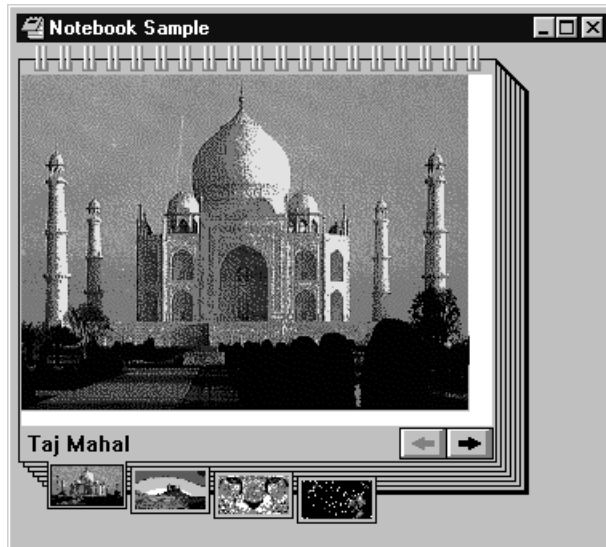
You can specify notebook styles during notebook creation to define the look and feel of the notebook, or you can use the default notebook styles provided by the Open Class Library. The default notebook styles are the following:

Style	Description
INotebook::backPagesBottomRight	Simulates recessed pages along the right and bottom edges of the notebook. This function is ignored in the Windows tab control implementation.
INotebook::majorTabsRight	Places major tabs on the right side of the notebook. For the Windows tab control, all tabs are minor and are at the top of the control.
INotebook::solidBinding	Binding is solid. This function is ignored in the Windows tab control implementation.
INotebook::squareTabs	Tabs are square-shaped in the PM-compatible control, rounded in the Windows tab control.
INotebook::statusTextLeft	Status text is left-justified. This function is ignored in the Windows tab control implementation.
INotebook::tabTextCenter	Tab text is centered.
IWindow::visible	Notebook is visible.

The following figure shows the appearance of the Windows native notebook control created using the default notebook styles. This notebook is from the Notebook Sample.



The next figure shows the appearance of the PM-compatible notebook control created using the default notebook styles and `INotebook::pmCompatible`. This notebook is also from the Notebook Sample.



The notebook control resembles a real notebook in its general appearance. For example, the notebook has a *binding* that, along with recessed pages on the right and bottom edges, gives the notebook a three-dimensional appearance. The binding is solid and placed on the left side, using the default INotebook styles `backPagesBottomRight` and `solidBinding`.

In the bottom right corner of the notebook are the *page buttons*. These buttons are for bringing one page of the notebook into view at a time. They are a standard component provided with every notebook. Your application can change the default width and height of the page buttons using `INotebook::setPageButtonSize`. However, you cannot query the size of the page buttons.

Selecting the *forward page button* (the arrow pointing to the right) causes the next page to be displayed; selecting the *backward page button* (the arrow pointing to the left) causes the previous page to be displayed.

To the left of the page buttons when using the default notebook style is the *status line*, which enables your application to provide information to the user about the page currently displayed. The notebook does not supply any default text for the status line. You are responsible for using `INotebook::PageSettings::setStatusText` or `INotebook::setStatusText` to associate a text string with the status line of each page. By default, the text in the status line is left-justified.

The page buttons are always located in the corner where the recessed edges of the notebook intersect. These recessed edges are called the *back pages*. The default notebook's back pages intersect in the bottom right corner, which means the recessed pages are on the bottom and right edges.

The back pages are important because their intersection determines where the *major tabs* can be placed, which in turn determines the placement of the binding and the *minor tabs*. You can use major and minor tabs to organize related pages into sections; minor tabs define subsections within major tab sections. The content of each section has a common theme, which is represented to the user by a tabbed divider, similar to a tabbed page in a notebook.

The default style, `INotebook::majorTabsRight`, specifies that major tabs, if used, are placed on the right side of the notebook. This is the default placement when the

back pages intersect at the bottom right corner of the notebook. The binding is located on the left because it is always located on the opposite side of the notebook from the major tabs.

Minor tabs are always placed perpendicular to the major tabs, based on the intersection of the back pages and the major tab placement.

Minor tabs are displayed only as the associated major tab page is selected or if the notebook has no major tab pages.

The default shape of the tabs used on notebook divider pages is square. You can change the default width and height of the major and minor tabs using `INotebook::setMajorTabSize` and `INotebook::setMinorTabSize`, respectively.

A notebook tab may contain either text or a bitmap. You can place text on a tab using `INotebook::PageSettings::setTabText` or `INotebook::setTabText`. Use `INotebook::PageSettings::setTabBitmap` or `INotebook::setTabBitmap` to place a bitmap on a tab. You cannot position a bitmap on a tab using the default support because the bitmap stretches to fill the rectangular area of the tab. However, you may use the owner draw support to control the positioning and drawing of the bitmap on a tab.

WIN

The Windows tab control has the following behavior:

- The `INotebook`'s implementation of the Windows tab control only supports major tab pages. However, it converts 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 left. Tab text alignment settings are ignored.
- The Windows tab control automatically sizes the tabs, unless you specify a size via `INotebook::setMajorTabSize`.
- 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`, during `INotebook` construction to enable multiple row tab support.
- The Windows tab control supports owner drawing of the tabs. However, you must specify the notebook style, `INotebook::handleDrawTabs`, 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.

RELATED CONCEPTS

- “Chapter 14. Notebooks” on page 203
- “Tab Controls” on page 214
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

RELATED TASKS

- “Create a Notebook”
- “Specify Notebook Styles” on page 209
- “Add Pages to the Notebook” on page 211
- “Remove Notebook Pages” on page 213
- “Change Notebook Colors” on page 214
- “Create a Tab Control” on page 216
- “Create an OS/2 Warp 4.0 Style Notebook” on page 219
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

Create a Notebook

Use the `INotebook` class to create and manage the notebook control window. You can create an object of this class using one of the following constructors:

- ```
INotebook(unsigned long windowId,
 IWindow* parent,
 IWindow* owner,
 const IRectangle& initial = IRectangle(),
 const Style& style = defaultStyle());
```
- ```
INotebook( unsigned long windowId,
           IWindow*     parentAndOwner );
```
- ```
INotebook(const IWindowHandle& handle);
```

Only the first of the three constructors creates a new notebook control. This constructor accepts a numeric identifier for the notebook, a pointer to a window object for its parent window, and a pointer to a window object for its owner window. You can optionally specify the position, size, and styles of the notebook.

The next two constructors wrap an existing notebook control. The first of these two constructors is designed to wrap a notebook control that is loaded as a dialog resource. It accepts a numeric identifier for the notebook, and a pointer to a window object for its parent and owner window. The last of these two constructors is designed to wrap an existing notebook control. It accepts the existing notebook's window handle.

Version 6 of the Hello World application creates a notebook as a private data member, `helloSettingsNotebook`, from a derived `IFrameWindow` class, called `ANotebookWindow`. The `helloSettingsNotebook` object is initialized on the `ANotebookWindow` constructor using the following `INotebook` constructor from the `anotebw6.cpp` file:

```
ANotebookWindow :: ANotebookWindow(unsigned long windowId,
 AHelloWindow * ownerWnd)
: IFrameWindow(windowId, IWindow::desktopWindow(),
 ownerWnd, IRectangle(),
 classDefaultStyle |
```

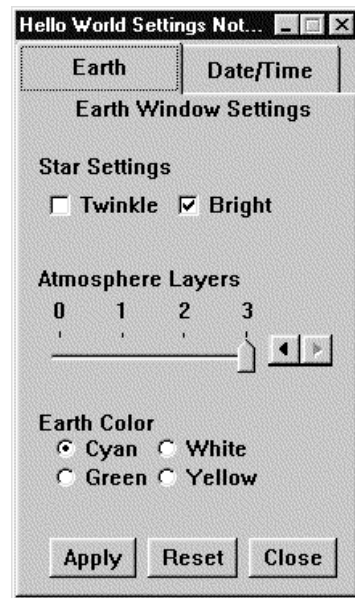
```

 dialogBackground |
 dialogBorder)
, helloSettingsNotebook(WND_NOTEBOOK, this, this)
// ...
{
 // ...
}

```

This constructor creates the notebook as a child window of the ANotebookWindow object and uses the default style.

The following figure shows the Hello World version 6 notebook control:



#### RELATED CONCEPTS

- “Chapter 14. Notebooks” on page 203
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

---

## Specify Notebook Styles

You can specify notebook styles during notebook creation to define the look and feel of the notebook. The Open Class Library provides notebook styles so that your application can specify or change the notebook’s styles.

When you specify an INotebook style on the notebook constructor, ensure that no conflicts occur. Many of the style choices are not independent from one another.

If you specify more than one style bit, you must use a bitwise OR operator (|) to combine them.

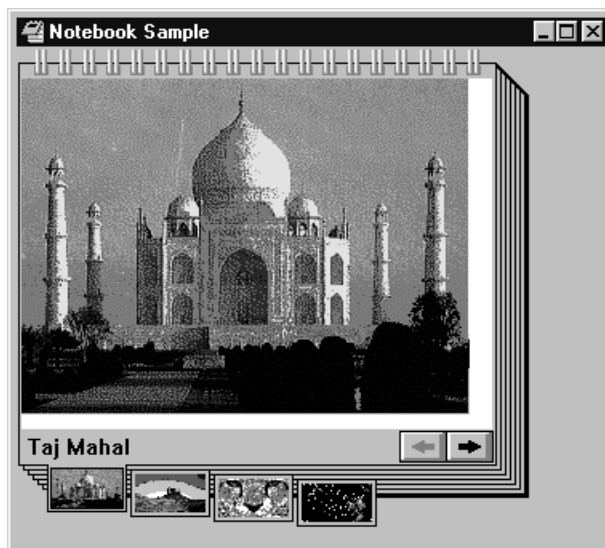
If you want to specify notebook styles other than the default when you are creating a notebook, create an object of the INotebook class, initialize it, and pass a reference to it on the constructor that accepts a style parameter. For example in the Notebook Sample .cpp file, we do the following:

```

ABitmapNotebook :: ABitmapNotebook(unsigned long windowId)
 :IFrameWindow(windowId)
 ,reslib()
 ,tabSize(50,50)
 ,tab1(reslib.loadBitmap(BM_PAGE1, tabSize))
 ,tab2(reslib.loadBitmap(BM_PAGE2, tabSize))
 ,tab3(reslib.loadBitmap(BM_PAGE3, tabSize))
 ,tab4(reslib.loadBitmap(BM_PAGE4, tabSize))
 ,notebook(WND_NB, this, this, IRectangle()
 ,_INotebook::defaultStyle()
 &_INotebook::majorTabsRight
 |_INotebook::majorTabsBottom
 |_INotebook::spiralBinding)
 ,viewport1(WND_VP1, ¬ebook, ¬ebook)
 ,viewport2(WND_VP2, ¬ebook, ¬ebook)
 ,viewport3(WND_VP3, ¬ebook, ¬ebook)
 ,viewport4(WND_VP4, ¬ebook, ¬ebook)
 ,bitmap1(BM_PAGE1, &viewport1, &viewport1, BM_PAGE1)
 ,bitmap2(BM_PAGE2, &viewport2, &viewport2, BM_PAGE2)
 ,bitmap3(BM_PAGE3, &viewport3, &viewport3, BM_PAGE3)
 ,bitmap4(BM_PAGE4, &viewport4, &viewport4, BM_PAGE4)
{
 // ...
}

```

The notebook created using the preceding statements has a spiral binding and tabs on the bottom:



You can create a notebook, specify its style, and change the size of major tabs and minor tabs with the following statements:

```

INotebook
*noteBook = new INotebook(ID_NOTEBOOK,
 this,
 this,
 IRectangle(),
 (INotebook::spiralBinding |
 INotebook::backPagesTopRight |

```

```

 INotebook::majorTabsRight |
 INotebook::statusTextLeft |
 IWindow::visible));
pnoteBook->setMajorTabSize(ISize(60, 30));
pnoteBook->setMinorTabSize(ISize(80, 40));

```

The notebook created using the preceding statements has a spiral binding, back pages on the right and top edges, major tabs on the right, status text that is left-justified, and is visible.

If you do not code the style on the first constructor, the static function, `INotebook::defaultStyle` determines the style. Initially, this function returns the default style, `INotebook::classDefaultStyle`. Use the function `INotebook::setDefaultStyle` to change the notebook's default style setting.

Once you create your notebook, you can change the various styles using member functions provided by `INotebook` that change the binding, orientation, tab shape, and tab text alignment of the notebook.

#### RELATED CONCEPTS

“Chapter 14. Notebooks” on page 203  
 “Chapter 1. Windows” on page 1  
 “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Use Notebooks” on page 204  
 “Create a Notebook” on page 208  
 “Write an Event Handler” on page 25  
 “Task and Samples Cross-Reference Table” on page 445

---

## Add Pages to the Notebook

A page setting object lets the user change and set information about pages in a notebook. Use the `INotebook::PageSettings` and `INotebook::PageSettings::Attribute` classes to create each page of the notebook.

Use the nested class `INotebook::PageSettings` to describe the characteristics of a notebook page before you add it to the notebook. An `INotebook::PageSettings` object defines page attributes like the use of major or minor tabs, automatic sizing behavior, the text or bitmap for the tabs, and the text for the status area.

Use `INotebook::PageSettings::Attribute` to specify the attributes of a page. This class is similar to a style class and has constant values defined for its values.

When inserting new pages into a notebook, carefully consider how your user will expect those pages to be organized. For an existing notebook especially, the underlying hierarchy must be observed when inserting new pages to provide efficient organization and navigation through the information contained in the notebook.

Use the following list of add functions to insert new pages into the notebook:

- `INotebook::addFirstPage()`
- `INotebook::addLastPage()`
- `INotebook::addPageAfter()`

- `INotebook::addPageBefore()`

All of these functions contain an optional parameter, `pageWindow`, that you can use to specify a page window to associate to a notebook page. A *page window* is an application window that you associate with a page of a notebook.

After a page is inserted into a notebook, you must facilitate the display of the information for this page when it is brought to the front of the notebook. The notebook provides a top page area in which the application can display page windows for the topmost page. For each inserted page, the application must associate the handle of a page window that is to be invalidated when the page is brought to the top of the book. You can associate the same handle with different pages.

The page window may be an `IFrameWindow` that you create from a dialog template, or it may be one of the `ICanvas` derived classes. You can associate a page window to a page of a notebook when you add the page to the notebook, or you can specify it later using `INotebook::setWindow()`. You determine the best time to associate your application window, immediately or delayed, based upon the resource requirements of your notebook.

You can define page windows using different window types, but we recommend using one of the canvas classes. When using a canvas class for your notebook page, you may need to set the size of the pages as in the following example:

```
page1.setSize(page1.minimumSize());
```

Use `IPageHandle` to identify a page once it is in the notebook. An `IPageHandle` is a wrapper class for the numeric identifier assigned to a page when you add it to the notebook. You use `IPageHandle` on most operations affecting the characteristics of the page and to remove the page from the notebook.

The following sample comes from the Hello World version 6 `anotebw6.cpp` file:

```
ANotebookWindow :: ANotebookWindow(unsigned long windowId,
 AHelloWindow * ownerWnd)
// ...
{
 // ...
 // Create the notebook page settings with attributes for a
 // major tab and automatic page size.
 // Set tab text to the Earth tab text from the resource
 // file.
 // Use the earthPage canvas and the notebook settings to
 // add a page to the beginning of the notebook.
 // Reset the tab text to the date and time tab text from
 // the resource file.
 // Use the dateTimePage canvas and the notebook settings
 // to add a page to the end of the notebook.
 // The Motif release of the User Interface Class Library
 // automatically sizes the major tab. For other releases,
 // manually resize the tabs.
 INotebook::PageSettings
 helloSettings(INotebook::PageSettings::majorTab|
 INotebook::PageSettings::autoPageSize);
 helloSettings.setTabText(IResourceId(STR_EARTHTAB));
 helloSettingsNotebook.addFirstPage(helloSettings,&earthPage);
 helloSettings.setTabText(IResourceId(STR_DATETIMETAB));
 helloSettingsNotebook.addLastPage(helloSettings,&dateTimePage);
 // ...
}
```



Each notebook page window, in this case, is a multicell canvas. Hello World version 6 creates a PageSettings object, `helloSettings`, to contain the specifications for each notebook page. The object is created with the `INotebook::PageSettings::Attributes` that specify major tabs and automatic page sizing. You use this PageSettings object to set the text in the tabs with the `setTabText()` member function.

To create the first page, use `INotebook::addFirstPage()` function. This function takes a PageSettings object and the page window that you want to associate with this page. The Hello World Version 6 sample uses `helloSettings` for the PageSettings object, and a reference to `earthPage` as the page window, to create its first page.

To create the next page, use `INotebook::addLastPage()`. If you wanted to add more pages to Hello World version 6, use `addLastPage()` again to append pages to the end of the notebook.

#### RELATED CONCEPTS

“Chapter 14. Notebooks” on page 203

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Use Notebooks” on page 204

“Create a Notebook” on page 208

“Specify Notebook Styles” on page 209

“Remove Notebook Pages”

“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Remove Notebook Pages

You can remove pages from the notebook by supplying the `IPageHandle` that was returned when the page was created.

Use the following `INotebook` functions to remove notebook pages:

#### **removePage**

Accepts a handle directly or determines the handle from an instance of an `INotebook::Cursor`.

#### **removeAllPages**

Removes multiple pages of a notebook.

#### **removeTabSection**

Removes the pages associated with a major or minor tab section.

#### RELATED CONCEPTS

“Chapter 14. Notebooks” on page 203

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Use Notebooks” on page 204  
“Create a Notebook” on page 208  
“Add Pages to the Notebook” on page 211  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

---

## Change Notebook Colors

Your application can tailor the color of almost any part of the notebook. Use the various color functions to change the colors of a notebook.

When you change the color in a control area, the presentation system passes this color change request to all the children of the control. This causes a child window with the same color control area to change to the new specified color if its control area has not been explicitly set. Therefore, changing colors in the notebook can cause changes to the page windows on the notebook.

Use the following INotebook functions to change the notebook’s color:

- setPageBackgroundColor()
- setMajorTabBackgroundColor()
- setMinorTabBackgroundColor()
- setMajorTabForegroundColor()
- setMinorTabForegroundColor()
- setBackgroundColor()
- setForegroundColor()

### RELATED CONCEPTS

“Chapter 14. Notebooks” on page 203  
“Chapter 1. Windows” on page 1  
“Chapter 3. Events and Event Handlers” on page 19

### RELATED TASKS

“Use Notebooks” on page 204  
“Create a Notebook” on page 208  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

---

## Tab Controls

The ITabControl class gives you another option to implement a notebook-like dialog. It creates a *tab control* similar to the native Windows tab control.

### AIX

AIX does not support the ITabControl class.

### OS/2

The OS/2 Warp 4 operating system supports the ITabControl class, but not OS/2 Warp Server 4. This control uses the BKS\_TABBEDDIALOG style on the PM notebook.

A tab control is a window that appears as a stack of tabbed pages (the tabs located at the top of the pages) with an set of optional push buttons.

#### WIN

You can use an ITabControl object as the client window of an IFrameWindow to create a tab control dialog.

#### The ITabPage Class

An ITabControl object manages a set of ITabPage objects and an optional set of IPushButton objects. You must create an ITabPage object or an object of a derived class for each page in the tab control. A tab page is a logical representation of a page in a tab control and has an associated page window and optional push buttons. A tab page object is not a wrapper for a platform window as are IWindow derived objects.

Each tab page has a style and either text or a bitmap to be displayed on the corresponding tab in the tab control. You can specify the text and bitmap for the tab page when constructing the ITabPage object, or by calling ITabPage::setTabBitmap or setTabText.

#### WIN

In the Windows environment, tab pages can have both text and a bitmap displayed on the tab simultaneously.

#### OS/2

In the OS/2 environment, you can assign both text and a bitmap for a tab, but OS/2 only displays the last item set.

#### Optional Push Buttons

The *optional push buttons* in the tab control appear in a button area below the tab pages, or, in the OS/2 environment, the button area can optionally appear above the tab pages. You can specify buttons on a per page basis as well as specifying a set of common buttons. ITabControl displays the set of common buttons whenever the top page has no page specific buttons. You must manage common buttons via ITabControl member functions and manage page specific buttons via ITabPage member functions.

You can add one of the five predefined push buttons provided by the EButton enumeration as common or page specific buttons. You can use the same buttons in multiple page button sequences as well as the common button sequence. This allows maximum flexibility in tailoring the buttons that appear as the user selects each tab page. For example, one tab page may have OK, Cancel, and Help buttons while all other pages have OK, Default, Cancel, and Help buttons. You would add the OK, Default, Cancel, and Help buttons as common push buttons and add the OK, Cancel, and Help buttons as page push buttons for the unique page.

You can also create your own IPushButton objects for use as tab control buttons. You must use the window returned by ITabControl::buttonParent as the parent and owner of any IPushButton objects you construct for use as either common buttons or page buttons.

## Command Handling

To process the command events generated by tab control buttons, attach a handler derived from  `ICommandHandler`  to the tab control or a window in its owner window chain.

### RELATED CONCEPTS

“Chapter 14. Notebooks” on page 203

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

### RELATED TASKS

“Create a Tab Control”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

## Create a Tab Control

WIN

OS/2

The  `ITabControl`  class gives you another option to implement a settings or properties dialog.  `ITabControl`  creates a window that appears as a stack of tabbed pages with an optional button area.

For OS/2, this control uses the  `BKS_TABBEDDIALOG`  style on the PM notebook. On the Windows operating system, this is the native tab control with some additional drawing to emulate the Windows property sheet.

You can use an  `ITabControl`  object as the client of an  `IFrameWindow`  to create a tab control dialog.

An  `ITabControl`  object manages a set of  `ITabPage`  objects and an optional set of  `IPushButton`  objects. You must create an  `ITabPage`  object or an object of a derived class for each page in the tab control. A tab page is a logical representation of a page in a tab control and has an associated page window and optional push buttons. A tab page object is not a wrapper for a platform window as are  `IWindow`  derived objects.

Each tab page has a style and either text or a bitmap to be displayed on the corresponding tab in the tab control. You can specify the text and bitmap for the tab page during  `ITabPage`  construction or by using one of the  `ITabPage`   `Tab Text & Bitmap`  member functions. In the Windows environment, tab pages can have both text and a bitmap displayed on the tab simultaneously. In the OS/2 environment, you can assign both text and a bitmap for a tab, but only the last item set is displayed.

The optional push buttons in the tab control appear in a button area below the tab pages, or, in the OS/2 environment, the button area can optionally appear above the tab pages. You can specify buttons on a per page basis as well as specifying a set of common buttons.  `ITabControl`  displays the set of common buttons whenever the top page has no page specific buttons. You must manage common buttons via  `ITabControl`  member functions and manage page specific buttons via  `ITabPage`

member functions. To process the command events generated by tab control buttons, create a handler of an  `ICommandHandler`  derived class and attach it to either the tab control or frame window.

You can add one of the five predefined push buttons provided by the  `EButton`  enumeration as common or page specific buttons. You can use the same buttons in multiple page button sequences as well as the common button sequence. This allows maximum flexibility in tailoring the buttons that appear as the user selects each tab page. For example, one tab page may have **Ok**, **Cancel**, and **Help** buttons while all other pages have **Ok**, **Default**, **Cancel**, and **Help** buttons. You would add the **Ok**, **Default**, **Cancel**, and **Help** buttons as common push buttons and add the **Ok**, **Cancel**, and **Help** buttons as page push buttons for the unique page.

You can also create your own  `IPushButton`  objects for use as tab control buttons. You must use the window returned by  `buttonParent`  as the parent of any  `IPushButton`  objects you construct for use as either common buttons or page buttons.

To process the command events generated by tab control buttons, you must attach a handler derived from  `ICommandHandler`  to the tab control or frame window.

In the following sample, from the Tab Control Sample,  `TabTest`  is our main window class. An  `ITabControl`  object is constructed and set as the frame window's client. Three different  `ITabPage`  derived objects are constructed. These three tab pages are then added to the tab control. Help is initialized for the application and a command handler is constructed and attached to the tab control to handle command events from the tab control push buttons.

The following code is from  `tabctl.cpp` :

```
TabTest :: TabTest(unsigned long windowId)
: IFrameWindow(windowId
 ,classDefaultStyle
 |shellPosition
 |dialogBackground
 |dialogBorder)
, tabControl(WND_TABCONTROL,
 this,
 this,
 IRectangle(),
 ITabControl::defaultStyle()
 | ITabControl::buttons)
, enrollPage(this)
, dataPage(this)
, helpWindow(HELP_TABLE,
 this)
, tabCommandHandler(this)
{
 // Add the three tab pages to the tab control.
 tabControl.addLastPage(&enrollPage);
 tabControl.addLastPage(&dataPage);
 tabControl.addLastPage(&photoPage);
 // Add three common buttons to the tab control.
 // These push buttons are displayed for pages which
 // don't have any page buttons set.
 tabControl.addCommonButton(ITabControl::kOk, true);
 tabControl.addCommonButton(ITabControl::kCancel);
 tabControl.addCommonButton(ITabControl::kHelp);
 // Set the application icon.
 // Set the tab control as the frame's client window.
 // Size the client area to the tab control's minimum size
 // Size the frame window to the client size.
```

```

setIcon(WND_MAIN);
setClient(&tabControl);
IRectangle clientRect = clientRectFor(rect());
clientRect.sizeTo(tabControl.minimumSize());
moveSizeToClient(clientRect);
// Start handling command events from the tab control
// buttons. You can attach the command handler to either
// the tab control or the frame. It is attached to the tab
// control here.
tabCommandHandler.handleEventsFor(&tabControl);
// Add the help library to the help window using
// addLibraries(). Set the help window title from a
// string in the resource file.
try
{
 helpWindow.addLibraries("tabctl.hlp");
 helpWindow.setTitle(STR_HTITLE);
}
catch(...)
{
 IMessageBox
 msgBox(this);
 msgBox.setTitle(STR_HELP_NOT_FOUND_TITLE);
 msgBox.show(STR_HELP_NOT_FOUND, IMessageBox::information);
}
}

```

In the `tabctl.hpp`, we define our class as follows:

```

class TabTest : public IFrameWindow
{
public:
 // Constructor
 // - constructs the object with a window ID
 TabTest(unsigned long windowId);
 // Destructor
 virtual
 ~TabTest() { };
 // Accessors
 // - These functions provide a means of getting and
 // settings the accessible attributes of instances
 // of this class:
 // sponsor - Returns the name of the sponsor.
 // setSponsor - Sets the name of the sponsor.
 // cheetahIndex - Returns the selected cheetah index
 // from enroll page
 IString
 sponsor() const;
 TabTest
 &setSponsor(const IString& currentSponsor);
 unsigned long
 cheetahIndex() const;
private:
 ITabControl
 tabControl;
 EnrollPage
 enrollPage;
 DataPage
 dataPage;
 PhotoPage
 photoPage;
 TabCommandHandler
 tabCommandHandler;
 IHelpWindow

```

```

 helpWindow;
 IString
 sponsorName;
};

```

We then go on to define and implement our accessor functions and various pages. The tab control uses delayed page window construction to allow for a user's choice. They are constructed only when needed by the application. Each page can contain a variety of controls.

#### RELATED CONCEPTS

"Tab Controls" on page 214  
 "Chapter 14. Notebooks" on page 203  
 "Chapter 1. Windows" on page 1  
 "Chapter 3. Events and Event Handlers" on page 19

#### RELATED TASKS

"Use Notebooks" on page 204  
 "Create a Notebook" on page 208  
 "Write an Event Handler" on page 25  
 "Task and Samples Cross-Reference Table" on page 445

---

## Create an OS/2 Warp 4.0 Style Notebook

### OS/2

You cannot create an OS/2 Warp 4.0 style notebook directly using the INotebook class, but you can create a notebook with this style (using WinCreateWindow) and construct an INotebook object for it. The following example illustrates this method:

```

/*****
* Merlin Notebook Sample Program: This program creates a Merlin style
* notebook as a client window. The notebook is created via WinCreateWindow
* and an object is then constructed for it using an INotebook wrapper
* constructor. The OS/2 Warp 4.0 Toolkit must be installed in order to
* implement this sample.
*****/
extern "C" {
 #define INCL_WIN
 #define INCL_WINSTDBOOK
 #include <os2.h>
}
#include <iapp.hpp>
#include <inotebk.hpp>
#include <iframe.hpp>
#include <istattxt.hpp>
#define WND_MAIN 1000
#define WND_NB 1010
#define WND_STATIC1 2110
#define WND_STATIC2 2120
#define WND_STATIC3 2130
#define WND_STATIC4 2140
/*****
* Class AMerlinNotebook – Sample application that constructs an OS/2
* Warp 4.0 BKS_TABBEDIALOG style notebook and wrappers it with an
* INotebook object.
*****/
class AMerlinNotebook : public IFrameWindow
{
public:
/*----- Constructor -----*/

```

```

 AmerlinNotebook(unsigned long windowId);
virtual
~AmerlinNotebook() { };
INotebook
*notebook;
IStaticText
*static1,
*static2,
*static3,
*static4;
};
/*****
* main() - Construct the frame window with a notebook client and run the *
* application. *
*****/
int main()
{
 AmerlinNotebook mainWindow(WND_MAIN);
 mainWindow.show();
 IApplication::current().run();
 return 0;
}
/*****
* AmerlinNotebook :: AmerlinNotebook - constructor *
* *
* Construct the IFrameWindow using the default style. *
* Create the BKS_TABBEDDIALOG notebook as a child window of the frame. *
*****/
AmerlinNotebook :: AmerlinNotebook(unsigned long windowId)
 :IFrameWindow(windowId)
{
 // Create a Merlin style notebook directly via WinCreateWindow.
 IWindowHandle hwndNotebook = WinCreateWindow(
 handle(), // parent
 WC_NOTEBOOK, // notebook class
 NULL, // window text
 BKS_TABBEDDIALOG // window style
 | BKS_MAJORTABTOP
 | WS_VISIBLE,
 0, 0, 0, 0, // window rectangle
 handle(), // owner
 HWND_TOP, // insertion order
 WND_NB, // window id
 NULL, // control data
 NULL); // presentation parms

 // Now create an INotebook object by using the INotebook wrapper
 // constructor.
 notebook = new INotebook(hwndNotebook);
 // Create some static text to use for the page windows.
 static1 = new IStaticText(WND_STATIC1, notebook, notebook);
 static2 = new IStaticText(WND_STATIC2, notebook, notebook);
 static3 = new IStaticText(WND_STATIC3, notebook, notebook);
 static4 = new IStaticText(WND_STATIC4, notebook, notebook);
 static1->setText("Page one");
 static2->setText("Page two");
 static3->setText("Page three");
 static4->setText("Page four");
 // Create a page settings object that specifies that pages should have a
 // major tab and pages should be automatically sized to fit the
 // notebook. For each page to be added to the notebook, set the tab text.
 INotebook::PageSettings pageSettings(
 INotebook::PageSettings::majorTab |
 INotebook::PageSettings::autoPageSize);
 pageSettings.setTabText("Page 1");
 notebook->addLastPage(pageSettings, static1);
 pageSettings.setTabText("Page 2");
 notebook->addLastPage(pageSettings, static2);
}

```



```

pageSettings.setTabText("Page 3");
notebook->addLastPage(pageSettings, static3);
pageSettings.setTabText("Page 4");
notebook->addLastPage(pageSettings, static4);
// Size the notebook and make it the client of the frame window.
notebook->sizeTo(ISize(400, 400));
setClient(notebook);
}

```

While the BKS\_TABBEDDIALOG notebook can be wrapped with an INotebook object, not all of the INotebook member functions are applicable to notebooks with this style. The following INotebook styles are not applicable to the Merlin style notebook:

- majorTabsLeft
- majorTabsRight
- polygonTabs
- roundedTabs
- squareTabs
- solidBinding
- spiralBinding
- statusTextLeft
- statusTextRight
- statusTextCenter
- tabTextLeft
- tabTextRight
- tabTextCenter

The following INotebook member functions should not be used with the Merlin style notebook:

- bindingsetBinding
- setPageButtonSize
- setStatusTextAlignment
- statusTextAlignment
- setMinorTabSize
- setTabShape
- tabShape
- setTabTextAlignment
- tabTextAlignment

#### **RELATED CONCEPTS**

“Chapter 14. Notebooks” on page 203

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### **RELATED TASKS**

“Use Notebooks” on page 204

“Create a Notebook” on page 208

“Specify Notebook Styles” on page 209

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445



---

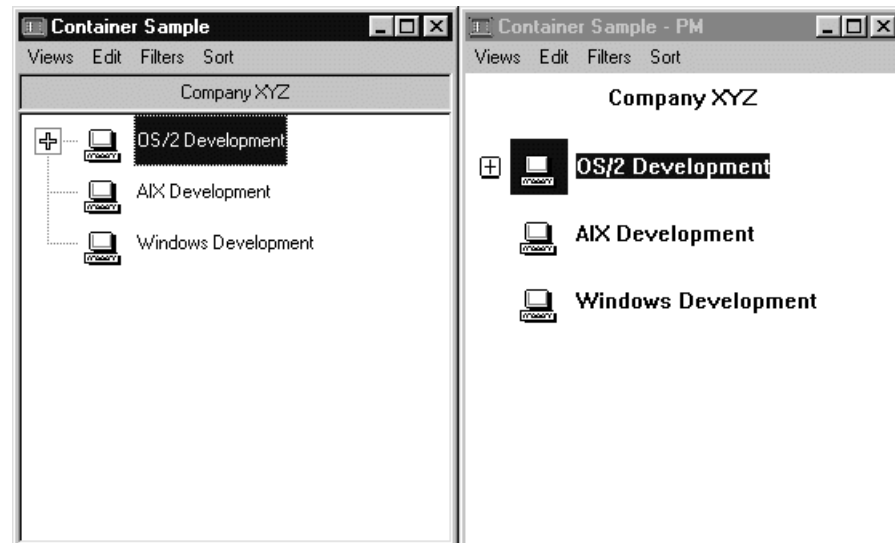
## Chapter 15. Containers

A *container* allows you to store, view, and manipulate non-window elements called objects. An *object* represents another container or an item such as an executable program, a word processing file, or a bitmap image. For example, the desktop window used by the graphical user interface of OS/2, Windows, and Motif are containers that hold program or data objects.

### WIN

The IContainerControl supports both an OS/2 Presentation Manager-compatible container and a container using the native Windows list view and tree view controls. By default, the IContainerControl class uses the Windows container style. The Windows container style gives more consistency between the look of the container in an IBM Open Class application and the Windows graphic user interface. On the other hand, the PM-compatible style allows you to easily port between OS/2 and Windows.

The following figure shows an example of two containers. The left application uses the Windows container style, while the one on the right uses the OS/2 container style:



### AIX

Due to the limited capabilities in Motif, the IBM Open Class does not support the following areas of IContainerControl and its related classes:

- Vertical split bar in the details view
- All emphasis, except selection and cursor emphasis
- The following ICnrHandler overrides will not be called:
  - deltaReached
  - windowScrolled
- Draw item (for example, owner draw)
- Scrolling functions, including:

- scroll
- scrollDetailsHorizontally
- scrollHorizontally
- scrollToObject
- scrollVertically
- IContainerControl constructors for creating an instance from an existing container control or a dialog template

### Views

Containers can display their objects in different views:

| View    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Text    | This view displays a text string representation of the objects of a container in a column.                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Name    | This view is an extension of the text view. It displays an icon representation of the object to the left of the text string.                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Icon    | This view displays an object's icon representation with its text string representation centered below the icon. The container positions the icon-text pair on the screen based on a pair of coordinates. Either the IContainerControl class, you, or the user may specify these coordinates                                                                                                                                                                                                                                                      |
| Tree    | This view displays a hierarchical view of a container's objects. A tree view lists its objects vertically. By default, the container draws a plus sign to the left of an object to indicate that the object (called the <i>parent</i> ) contains other objects (called <i>child objects</i> ). The user may <i>expand</i> a parent object to view its child objects. Conversely, a minus sign indicates that the user may <i>collapse</i> the object, hiding its child objects from view. You may only display child objects with the tree view. |
| Details | This view displays detailed information about each object of a container in a chart form. Each row contains the information about one object. Each column relates to one piece of data common to all the objects in the container.                                                                                                                                                                                                                                                                                                               |

### RELATED CONCEPTS

- "Chapter 23. Direct Manipulation" on page 313
- "Chapter 1. Windows" on page 1
- "Chapter 3. Events and Event Handlers" on page 19

### RELATED TASKS

- "Create and Use Containers" on page 225
- "Use Default Direct Manipulation" on page 316
- "Write an Event Handler" on page 25
- "Task and Samples Cross-Reference Table" on page 445

---

## Create and Use Containers

Use the `IContainerControl` class to create an instance of a container object. With this class, you can control, for example, the view of the objects inside the container. The following example shows one way to create a container:

```
IContainerControl cnrCtl(CNR_RESID, this, this);
```

Several styles are available for containers that you can use to manage such activities as multiple-selection and automatic positioning.

You can define the styles in the constructor, or you can use member functions to set the style required after you create an instance of the container object. An example of a style statement is highlighted in the following code:

```
cnrCtl = new IContainerControl (CNR_RESID, this, this);
cnrCtl->setExtendedSelection();
```

The `IContainerControl` supports both the OS/2 Presentation Manager-compatible container and the Windows container. The Windows container is a ported version of the PM container control. The Windows container is composed of the list view and tree view control which are native to the Windows environment. On all platforms, you can get PM style controls using the `IContainerControl::pmCompatible` option. On NT, you can also get the Windows list view control. This style of control is not available on OS/2 or AIX. The Windows list view control is the default selection in the Windows environment.

### RELATED CONCEPTS

“Chapter 15. Containers” on page 223

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

### RELATED TASKS

“Use Default Direct Manipulation” on page 316

“Task and Samples Cross-Reference Table” on page 445

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

### RELATED REFERENCES

`ICnrHandler`

`IContainerColumn`

`IContainerObject`

---

## Create Container Objects

Because a container has no meaning without its objects, use the `IContainerObject` class to create objects to put into it. At a minimum, an `IContainerObject` has an icon and a name.

The following is an example of an `IContainerObject` constructor:

```
IContainerObject
(const IString& string,
const IPointerHandle& iconHandle = 0);
```

To design your own objects for your applications, create a class that is derived from the `IContainerObject` class. If you use multiple inheritance, you must list the `IContainerObject` class first. To create a container object with department names,

addresses, and zip codes for your company, define this class as follows. This excerpt is from the Container Sample. You can find this sample in the samples/ioc/cnr directory:

```
class Department : public IContainerObject
{
public:
 Department(const IString& Name,
 const IPointerHandle& Icon,
 const IString& Code,
 const IString& Address);
 IString Code()
 const {return strCode;}
 IString Address()
 const {return strAddress;}
 void setCode (IString code)
 {strCode = code;}
 void setAddress (IString address)
 {strAddress = address;}
 virtual void handleOpen
 (IContainerControl* container);
private:
 IString strAddress;
 IString strCode;
};
```

The statements for a constructor definition are as follows:

```
Department :: Department(const IString& Name,
 const IPointerHandle& Icon,
 const IString& Code,
 const IString& Address):
 IContainerObject(Name, Icon),
 strCode (Code),
 strAddress (Address),
 {}
```

After you define the class, create an instance of an object using either of the following statement:

```
dept1 = new Department (
 "OS2 Development",
 IApplication::current().userResourceLibrary().loadIcon(IBMLOGO),
 "TWPD",
 "Building 71");
dept2=new Department(reslib.loadString(STR_ITEM_21),
 reslib.loadIcon(CLOGO),
 reslib.loadString(STR_ITEM_22),
 reslib.loadString(STR_ITEM_23));
```

You can find the Container Sample in the samples/ioc/cnr directory:

#### **RELATED CONCEPTS**

"Chapter 15. Containers" on page 223

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

#### **RELATED TASKS**

Container Sample

"Create and Use Containers" on page 225

"Add and Remove Container Objects" on page 227

"Share Objects Among Containers" on page 229

"Filter Container Objects" on page 230

"Sort Objects in a Container" on page 232

“Access Container Objects Using an Object Cursor” on page 233

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

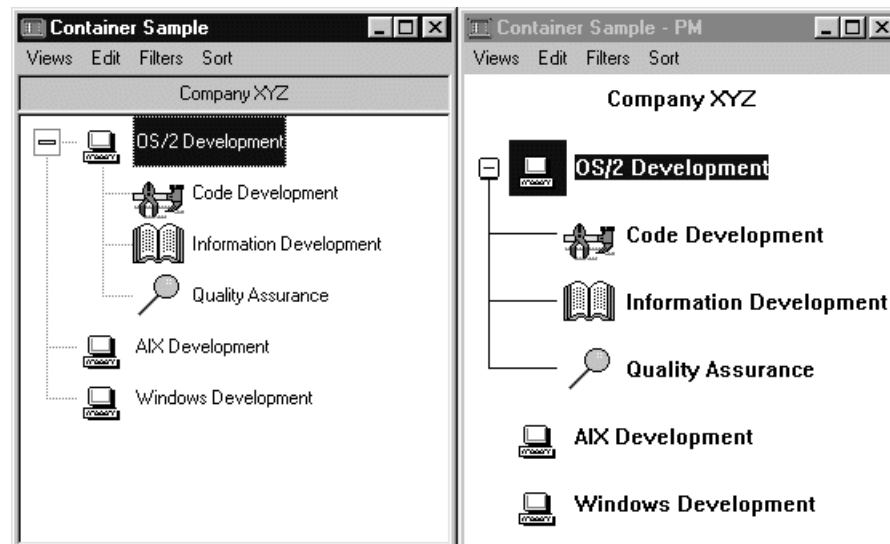
---

## Add and Remove Container Objects

After you create the objects and the container, add the objects to the container. The following statements from the Container Sample add objects to the container, `cnrCtl`. The first line adds an object, `dept1`. The next three lines add `dept2`, `dept3`, and `dept4` in a hierarchy under `dept1`. The last two lines add `dept5` and `dept6`.

```
cnrCtl->addObject(dept1); // Add Department 1 to container
cnrCtl->addObject(dept2,dept1); // Add Department 2 under Department 1
cnrCtl->addObject(dept3,dept1); // Add Department 3 under Department 1
cnrCtl->addObject(dept4,dept1); // Add Department 4 under Department 1
cnrCtl->addObject(dept5); // Add Department 5 to container
cnrCtl->addObject(dept6); // Add Department 6 to container
```

When you place the container in the client window and show the window and the container, you see a window like the one below.



You can find the source files of the Container Sample in the `samples/ioc/cnr` directory.

### Example of a Container Showing Objects in an Expanded Tree View

The window shows a tree view of the container’s objects. This view is discussed later.

#### WIN

Note that in these figures we show both the PM compatible version and the Windows native container.

You can also use the `ICnrAllocator` class to allocate a list of container items to be inserted into an `IContainerControl`. When you construct instances of this class, you can allocate memory from the container control for a specified number of objects with one call.

The `IContainerControl::addObjects` member function inserts all the initialized items of the allocator.

The following example shows how to use the `ICnrAllocator` class and the `IContainerControl::addObjects` member function:

```
/* ***** */
/* Define your derived IContainerObject class */
/* ***** */
class MyObject : public IContainerObject
{
public:
 MyObject(const IString& name) : IContainerObject(name) {}
 ~MyObject() {}
};
/* ***** */
/* Create a frame and a container */
/* ***** */
IFrameWindow frame(0x1300);
IContainerControl cnr(0x1400, &frame, &frame);
cnrCtl.showTextView();
/* ***** */
/* Create an allocator and allocate 10000 objects */
/* ***** */
ICnrAllocator allocator(10000, sizeof(MyObject));
/* ***** */
/* Initialize all 10000 */
/* ***** */
for(int i=0; i<10000; i++)
{
 new(allocator) MyObject("Peter");
}
/* ***** */
/* Add all the objects to the container */
/* ***** */
cnrCtl.addObjects(allocator);
```

By default, the container only removes objects when the container is deleted. It does not delete them. However, you can delete all objects in the container when the container is deleted by using the following code statement:

```
cnrCtl->setDeleteObjectsOnClose();
```

You can call `IContainerControl::deleteAllObjects` to delete all objects in a container. Specify the style `IContainerControl::noSharedObjects` when you create a container that does not share any objects with other containers. This increases the performance of the `IContainerControl::deleteAllObjects` member function.

The following example shows how to create a container with the `noSharedObjects` style:

```
/* ***** */
/* Create a container with the noSharedObjects style */
/* ***** */
IContainerControl* cnrCtl = new IContainerControl(0x1400, &frame,
 &frame, IRectangle(0,0,0,0),
 IContainerControl::defaultStyle() |
 IContainerControl::noSharedObjects);
```

#### RELATED CONCEPTS

“Chapter 15. Containers” on page 223

“Chapter 1. Windows” on page 1



“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

Container Sample

“Create and Use Containers” on page 225

“Create Container Objects” on page 225

“Share Objects Among Containers”

“Filter Container Objects” on page 230

“Sort Objects in a Container” on page 232

“Access Container Objects Using an Object Cursor” on page 233

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

IContainerControl

IContainerObject

---

## Share Objects Among Containers

You can also create objects and place them in multiple containers. The same object is then shared by two or more different containers.

In the Container Sample, dept2, dept3, and dept4 are in a hierarchy under dept1. We now want to create another container with only the main departments. This new container will then share dept1, dept5, and dept6 with the other container.

The following statements add three objects to a container, `cnrCt12`, that already exists in another container, `cnrCt1`. You can find this sample in the `samples/ioc/cnr` directory.

```
/*
/* Container with all departments */
/*
cnrCt1->addObject(dept1); // Add Department 1 to container
cnrCt1->addObject(dept2,dept1); // Add Department 2 under Department 1
cnrCt1->addObject(dept3,dept1); // Add Department 3 under Department 1
cnrCt1->addObject(dept4,dept1); // Add Department 4 under Department 1
cnrCt1->addObject(dept5); // Add Department 5 to container
cnrCt1->addObject(dept6); // Add Department 6 to container
/*
/* Container with main departments only */
/*
cnrCt12->addObject(dept1); // Add Department 1 to second container
cnrCt12->addObject(dept5); // Add Department 5 to second container
cnrCt12->addObject(dept6); // Add Department 6 to second container
```

Since the same object can exist in more than one container, the attributes of an object also reflect the state of that object. For example, an object can be visible in one container but hidden in another. You should consider the state of an object’s attribute in each container and the state of the attribute in each place the object resides.

The following container attributes can be modified:

- Visibility (PM-compatible container only)
- Cursored emphasis
- Selection emphasis
- In-use emphasis
- Refresh status

- Open status
- Direct edit status
- Expanded or collapsed state in tree view
- Target emphasis
- Source emphasis (PM-compatible container only)

Use the base container handler, `ICnrHandler`, to capture the event notifications provided by the container class. When the values of object attributes in the container change, these series of notifications are captured by the handler and routed to virtual functions within the handler.

For example, for both containers to reflect the same selection emphasis, you must attach an `ICnrHandler` to keep the objects in the same state in each container. Once both icons are selected, the same action is performed on both containers.

If you are performing multiple actions that cause the container to refresh, you can manipulate the refresh state so that the container will not repaint, as follows:

```

cnrCnt1.setRefreshOff();
:
:
cnrCnt1.setRefreshOn();
cnrCnt1.refresh();

```

#### RELATED CONCEPTS

“Chapter 15. Containers” on page 223

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

Container Sample

“Create and Use Containers” on page 225

“Create Container Objects” on page 225

“Add and Remove Container Objects” on page 227

“Filter Container Objects”

“Sort Objects in a Container” on page 232

“Access Container Objects Using an Object Cursor” on page 233

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

`IContainerObject`

---

## Filter Container Objects

You can filter objects in a container when using the PM-compatible container control. The container uses the `FilterFn` nested class to show a subset of the existing objects by filtering some of the objects.

#### WIN

The native Windows list view and tree view controls do not support filtering container objects.

To create a filter, do the following:

1. Define a class derived from the `FilterFn` class and override the member function `isMemberOf` to code the conditions of a valid object.

```

class SelectedObjectsFilter : public IContainerControl::FilterFn
{
virtual bool
 isMemberOf(IContainerObject* object,
 IContainerControl* container) const
 {
 return container->isSelected(object);
 }
};

```

If true is returned by the FilterFn derived class, the container object remains displayed in the container; if false, the object is hidden.

The isSelected member function returns true if the object has selection emphasis.

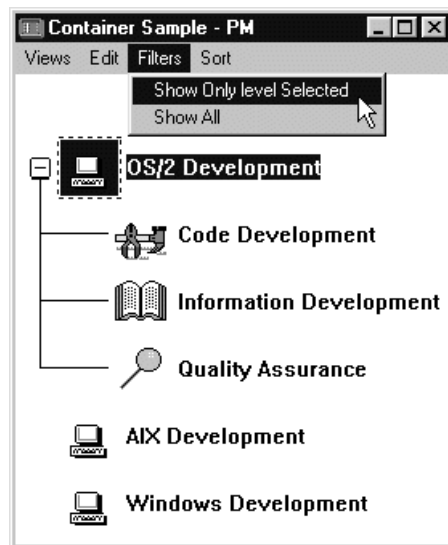
2. Call IContainerControl::filter. Use the following statements:

```

SelectedObjectsFilter selObjects;
cncctl->filter(selObjects);

```

**Before Filtering the Container Objects:**



**After Filtering the Container Objects:**



This output is from the Container Sample. You can find this sample in the samples/ioc/cnr directory.

#### RELATED CONCEPTS

"Chapter 15. Containers" on page 223

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

#### RELATED TASKS

Container Sample

"Create and Use Containers" on page 225

"Create Container Objects" on page 225

"Add and Remove Container Objects" on page 227

"Share Objects Among Containers" on page 229

"Sort Objects in a Container"

"Access Container Objects Using an Object Cursor" on page 233

"Write an Event Handler" on page 25

"Task and Samples Cross-Reference Table" on page 445

#### RELATED REFERENCES

IContainerControl

---

## Sort Objects in a Container

To sort objects in the container using their icon text, call `IContainerControl::sortByIconText`. You can also use a comparison function provided by your application. With an application comparison function, you can sort the objects in the container a variety of different ways. When you call the function `sortByIconText`, the container provides its own comparison function to do the sorting.

To provide your own sort behavior, do the following:

1. Define a class derived from `IContainerControl::CompareFn` and implement the function `isEqual`. The function `isEqual` should return an integer value that is

less than zero if the first object is less than the second object, zero if the first object is equal to the second object, or greater than zero if the first object is greater than the second object.

2. Create an object of the newly defined comparison class. Call the function `IContainerControl::sort`, and pass the comparison function object.

The following example sorts the objects in our container by two different criteria:

```
 case ID_SORT1:
 {
/*-----
| Sort the container based on the Icon Text.
| We call a container function to do this.
|-----*/
 pcnr->sortByIconText (true);
 break;
 }
 case ID_SORT2:
 {
/*-----
| Sort the container based on the Code Text.
| We must create an instance of the SortByCode class and pass this in.
|-----*/
 SortByCode sortByCode;
 pcnr->sort(sortByCode);
 break;
 }
}
```

To view the entire sample, see the `acnr.cpp` file in the `samples` directory.

#### RELATED CONCEPTS

“Chapter 15. Containers” on page 223

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Create and Use Containers” on page 225

“Create Container Objects” on page 225

“Add and Remove Container Objects” on page 227

“Share Objects Among Containers” on page 229

“Filter Container Objects” on page 230

“Access Container Objects Using an Object Cursor”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

`IContainerControl`

---

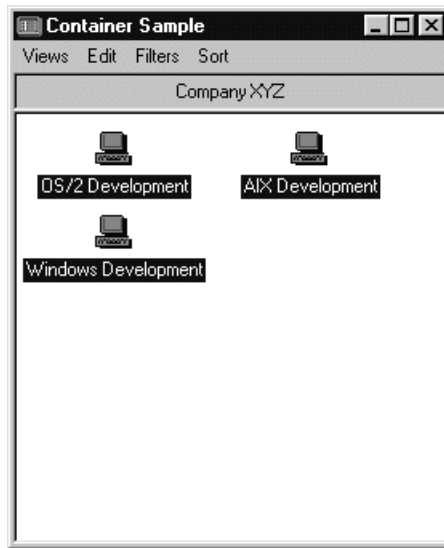
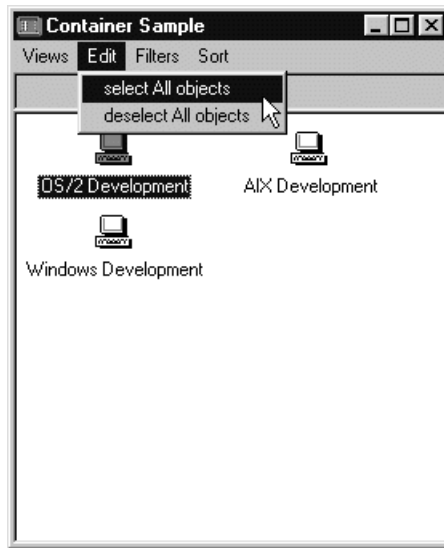
## Access Container Objects Using an Object Cursor

Use an object cursor to find all objects or find only those objects that meet a specific criteria.

The following example creates an `ObjectCursor` and uses it to select all container objects:

```
IContainerControl::ObjectCursor C01 (*cncCtrl);
for (C01.setToFirst(); C01.isValid(); C01.setToNext())
{
 cncCtrl->setSelected(cncCtrl->objectAt(C01));
}
```

The following figure shows the before and after result of setting the selection emphasis using an object cursor.



**RELATED CONCEPTS**

- "Chapter 15. Containers" on page 223
- "Chapter 1. Windows" on page 1
- "Chapter 3. Events and Event Handlers" on page 19

**RELATED TASKS**

- "Create and Use Containers" on page 225
- "Create Container Objects" on page 225
- "Add and Remove Container Objects" on page 227
- "Share Objects Among Containers" on page 229
- "Filter Container Objects" on page 230
- "Sort Objects in a Container" on page 232
- "Write an Event Handler" on page 25

“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

IContainerControl

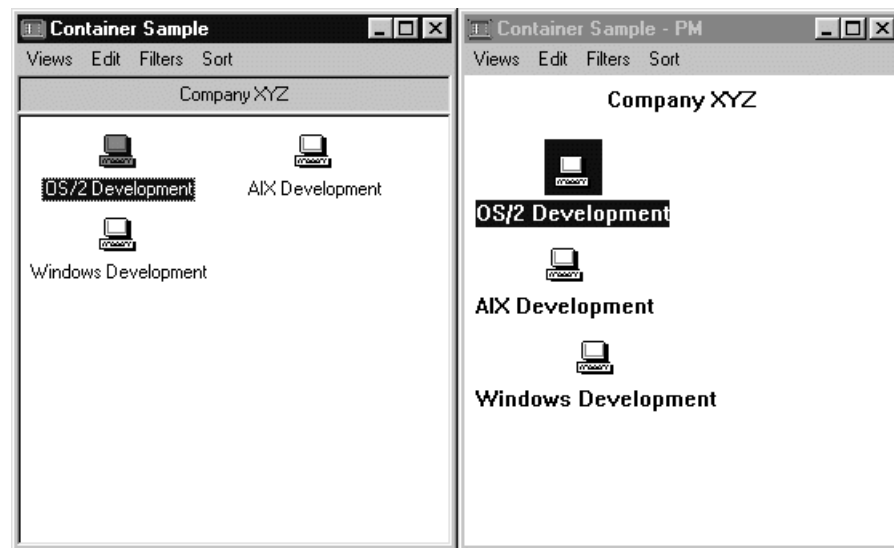
---

## Change Views in a Container

You can specify the view using a style on the constructor, or you can set it with a member function. For example, the following statement uses the member function that causes a container to display the icon view:

```
cnrCtl->showIconView();
```

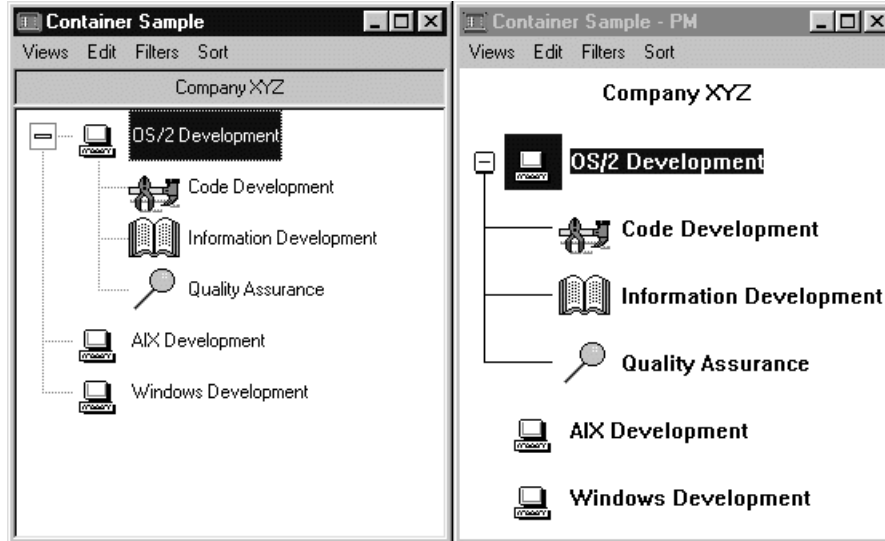
This statement provides the container view shown below:



The following statement provides the tree icon view:

```
cnrCtl->showTreeIconView();
```

The following figure shows a container with the tree icon view:



#### RELATED CONCEPTS

“Chapter 15. Containers” on page 223

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Define the Details View Using Container Columns”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Define the Details View Using Container Columns

The `IContainerColumn` class defines the information that is displayed for a given object when the container is in the details view. Only the items that you added with no parent display in the details view. You can use this class to set text in the heading of the columns, add horizontal and vertical separators by column, and align the column contents.

The following code is from the Container Sample. You can find this sample in the `samples/ioc/cnr` directory.

One way to create an instance of an `IContainerColumn` is for you to provide the offset of the object data to be displayed in the column and, optionally, the styles to be used for the heading and data.

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

#### > C++

ANSI C++ allows use of the `offsetof` macro only for structures and not classes. Use the following macro if your compiler raises an error for `offsetof`:

```
ICONTAINERCOLUMN_OFFSETOF
```

To create an instance of a container column, use the following statements:



```
colIcon = new IContainerColumn (IContainerColumn::isIcon);
colName = new IContainerColumn (IContainerColumn::isIconViewText);
colCode = new IContainerColumn (ICONTAINERCOLUMN_OFFSETOF(Department, strCode));
colAddress = new IContainerColumn (ICONTAINERCOLUMN_OFFSETOF(Department, strAddress));
```

In the previous example, colIcon, colName, colCode, and colAddress are defined as members of an IContainerColumn. The statements look like this:

```
private: //Define private information
 IContainerControl * cnrCtl;
 Department *dept1, *dept2, *dept3, *dept4, *dept5, *dept6 ;
 IContainerColumn *colIcon, *colName, *colCode, *colAddress;
 IMenuBar * menuBar;
```

After creating the container columns, you can add heading text to them using the following statements:

```
colIcon->setHeadingText("Icon");
colName->setHeadingText("Department Name");
colCode->setHeadingText("Code");
colAddress->setHeadingText("Address");
```

#### ► OS/2

When using the PM-compatible container you can use showSeparators to add a vertical separator next to a column or a horizontal separator under the heading text. The default adds both. To create only one of the separators, specify it in the member function statement. The following statements show examples of how to create separators:

```
 //Only Horizontal Separator
colIcon->showSeparators(IContainerColumn::horizontalSeparator);
 //Only Vertical Separator
colName->showSeparators(IContainerColumn::verticalSeparator);
colCode->showSeparators(); //both separator by default
colAddress->showSeparators(); //both separator by default
```

After you create the container columns, add them into the container using the following statements:

```
cnrCtl->addColumn(colIcon);
cnrCtl->addColumn(colName);
cnrCtl->addColumn(colCode);
cnrCtl->addColumn(colAddress);
```

The following shows an example of a details view of a container:



#### OS/2

When using the PM-compatible container control, you can use the following code statement to put a split bar in the details view by specifying the last column to be viewed in the left window and the location of the split bar in pixels. (This call is ignored on Windows when using a native container control and on AIX.)

```
cnrCtl->setDetailsViewSplit(colName, 350);
```

#### WIN

Separators and split bars are ignored in the native Windows list view and tree view containers. These options are supported by the PM control on both the Windows and OS/2 operating systems. The native control, however, provides dynamic sizing of all columns.

You can find the Container Sample in the `samples/ioc/cnr` directory.

#### RELATED CONCEPTS

“Chapter 15. Containers” on page 223

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

Container Sample

“Change Views in a Container” on page 235

“Use the Windows Native Container Control” on page 240

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Create a Pop-Up Menu in a Container

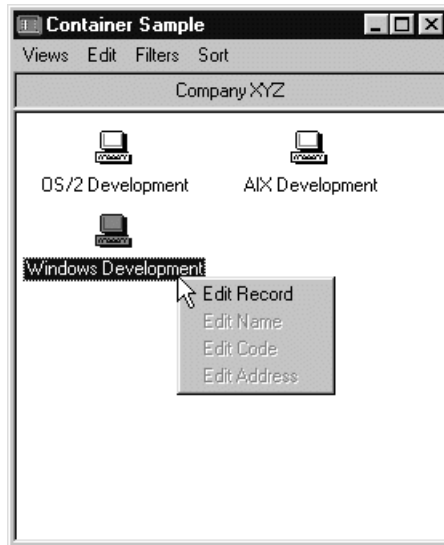
To create a pop-up menu in a container, create a class derived from `ICnrMenuHandler` and override the `makePopUpMenu`. The following statements in `acnr.hpp` create the class:

```
class ACnrMenuHandler: public ICnrMenuHandler
{
public:
/*----- Constructors/Destructor -----
| Construct the object in only one way:
| 1) Parameter for IContainerControl
|-----*/
 ACnrMenuHandler(IContainerControl* cnr) { pcnr = cnr; };
protected:
 virtual bool
 makePopUpMenu(IMenuEvent& event);
private:
 IContainerControl
 *pcnr;
};
```

After overriding the `makePopUpMenu` member function, you can add your own statements. The following statements in `acnr.cpp` create a pop-up menu displayed next to a container object with source emphasis:

```
bool ACnrMenuHandler::makePopUpMenu(IMenuEvent& event)
{
/*-----
| If a valid container object, continue
|-----*/
 if (popupMenuObject())
 {
/*-----
| Create a popup menu
| If not in details view, disable editing of Name, Code, and Address columns.
|-----*/
 IPopupMenu* popUp = new IPopupMenu(ID_POPMENU,
 event.window());
 if (!pcnr->isDetailsView())
 {
 popUp->disableItem(MI_EDNAME);
 popUp->disableItem(MI_EDCODE);
 popUp->disableItem(MI_EDADDRESS);
 }
 else
 {
/*-----
| Disable editing of the object
|-----*/
 popUp->disableItem(MI_EDRECORD);
 }
/*-----
| To avoid memory leaks, auto delete the C++ popup menu when the GUI popup
| window closes
| Show the popup menu at the current mouse position
| Visually indicate the container object is the source for this action
| Visually indicate the container object is the current cursor
|-----*/
 popUp->setAutoDeleteObject();
 pcnr->showSourceEmphasis(popupMenuObject());
 pcnr->setCursor(popupMenuObject());
 popUp->show(event.mousePosition());
 return true;
 }
 return false;
};
```

The following figure shows the pop-up menu in a container object:



#### RELATED CONCEPTS

"Chapter 15. Containers" on page 223

"Chapter 4. Menus" on page 55

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

#### RELATED TASKS

"Create and Use Containers" on page 225

"Write an Event Handler" on page 25

"Task and Samples Cross-Reference Table" on page 445

#### RELATED REFERENCES

IPopUpMenu

---

## Use the Windows Native Container Control

### WIN

This section lists the differences between the native Windows container control and the PM-compatible control.

The default IContainerControl wrappers the Windows native ListView and TreeView controls. The underlying controls are created when needed. If you create an IContainerControl in the default icon view, only a ListView control is created. A TreeView control is not created until the view is switched to a tree view or a child object is added. To optimize a tree view container, consider the following:

- Create the container in tree view and do not switch to a nontree view. This optimizes adding and removing objects because there is only one underlying control.
- Call IContainerControl::setDefaultStyle to set the initial value.

#### Constructors

For the native Windows control, you cannot use the following constructors:

```
IContainerControl (unsigned long id,
 IWindow* parentDialog);
IContainerControl (const IWindowHandle& handle);
```

### Member Functions

Source emphasis is not supported. This is true of both the Windows CUA control and the native Windows control.

There is no windows native multiple selection. `IContainerControl::multipleSelection` is the same as `extendedSelection` and `IContainerControl::verifyPointers` is ignored.

The following `IContainerControl` attributes are ignored:

- `readOnlyTitle`
- `detailsViewTitles`
- `handleDrawItem`
- `handleDrawBackground`
- `orderedTargetEmphasis`
- `mixedTargetEmphasis`
- `visibleTreeLine`

The following are not supported:

#### **showTextView**

Always flowed using the native container

#### **showNameView**

Non tree views are all flowed using the native container

#### **setTreeViewIndent**

Not supported using the native control

#### **setTreeItemIcons**

Not supported using the native control

#### **setTreeExpandIconSize**

Not supported using the native control

The Open Class Library does not support the following when using the native Windows container control:

- You cannot change the minimum distance required by the container for emphasis painting as line spacing is not supported.
- The filtering of container objects is not supported.
- The container title cannot be updated.
- Column titles are always visible.
- Scrolling in the tree view is not supported.
- You cannot change the direct manipulation target emphasis.
- The `HeadingStyle` is the same as `IContainerColumn::DataStyle`, must be text, and is always read-only.
- You cannot put an icon in a column heading.
- A column heading is always read only.
- You cannot filter out container columns.
- You cannot change a column's separator styles. Vertical column separators are not supported and horizontal column separators are required.
- You cannot change a column heading's style.

- You cannot change a column heading's attributes.
- Split bars are not supported in native containers. However, all columns are sizeable by the user.
- Data caching is not supported.
- The `handleDrawItem` and `handleDrawBackground` attributes are not supported.
- The `ICnrAllocator` class can be used but the performance advantage is not as great as when used with the `pmCompatible` style. The container objects must be inserted one at a time when using the native control.

Note that the native `ListView` control requires the first column in a details view to be the icon and text pair that is displayed in the other views. This column is sized to nonzero if either a `IContainerColumn` with

```
objectDataType=isIcon
```

or

```
objectDataType=isIconViewText
```

is added. Regardless of where they are added, these columns are displayed as the first column. If neither is added, the column size is set to zero but it is possible for the user to size this column using the mouse.

This is the only icon that may be displayed in the details view. If you add any other columns to display icons, the icons are not visible.

The native control allows selection only in this first column. If this column is sized to zero, objects in the container cannot be selected.

#### **RELATED CONCEPTS**

"Chapter 15. Containers" on page 223

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

#### **RELATED TASKS**

"Create and Use Containers" on page 225

"Write an Event Handler" on page 25

"Task and Samples Cross-Reference Table" on page 445

#### **RELATED REFERENCES**

`IContainerControl`

---

## Chapter 16. Toolbars

A *toolbar* is a window whose buttons represent tools, menu items, or actions. The toolbar can be located along the top, bottom, or sides of a frame window or can “float” and be positioned anywhere on your desktop. You can also have multiple toolbars with a variety of different toolbar buttons using text, bitmaps, or both.

You create toolbars with the `IToolBar` class. The `IToolBarContainer` class allows you to manage more than one toolbar. Toolbar buttons and combo boxes are probably the most common types of windows you will use in a toolbar. You can create toolbar buttons with the `IToolBarButton` class.

The `IToolBar` (and the `IToolBarContainer`) classes derive from `ISetCanvas`. Therefore, these toolbar classes can take advantage of the set canvas’s functionality:

- Add any object derived from `IWindow` to a toolbar.
- Specify the amount of space, or *pad*, between groups of buttons on a toolbar.
- Lay out your the buttons of your toolbar into *decks* like a set canvas.
- Create multiple toolbars that you can lay out and display like a set canvas.

### Positioning Toolbars

You may position your toolbar in one of two locations:

- *Along one edge of the client window.* The Open Class Library essentially treats the toolbar as a *frame extension* in this case. (Frame extensions are rectangular areas whose size and position the frame window will manage.) The default location for a toolbar is along the top of the client window.
- *Floating.* The `IToolBar` class puts your toolbar into the client area of a new frame window separate from the one the toolbar is associated with.

#### OS/2

The user can “pin” the frame of the floating toolbar to the associated frame window so it moves with that frame window.

#### AIX

There are restrictions on changing the position of a toolbar. For example, you cannot change between a floating and non-floating toolbar. Also there are cases when you cannot change the edge of the client area that the toolbar is positioned next to when displaying multiple toolbars. See the `IToolBar::isMoveValid` function for more information.

You can make the toolbar invisible by using the `IToolBar::hide` function. Use the `show` function to make the toolbar visible again.

### Non-Floating Toolbars

You may put as many toolbars you want in a frame extension. Suppose that you create a toolbar with the `IToolBar` class that you intend to position along an edge of a client window. The `IToolBar` class will perform the following:

1. Create an `IToolBarContainer` object
2. Put the `IToolBarContainer` object into the intended frame extension
3. Make your toolbar a child of the `IToolBarContainer` object

If an `IToolBarContainer` object already exists in the intended frame extension, the `IToolBar` class makes your toolbar a child of the existing `IToolBarContainer` object. Since the `IToolBarContainer` class inherits from the `ISetCanvas` class, you may treat the children of the `IToolBarContainer` class (your toolbars) like a set canvas.

### Floating Toolbars

When you create a floating toolbar, or change the location of a toolbar to floating, the `IToolBar` creates an `IToolBarFrameWindow` to contain your floating toolbar. You can only put one toolbar in a `IToolBarFrameWindow` object.

When the user closes a floating toolbar from the close item on the floating frame window's system menu, the toolbar is hidden. This is done through a call to `IToolBar::hide`. Consequently, neither the floating frame, the toolbar, or the windows within the toolbar are destructed. You can re-show the toolbar with a call to `IToolBar::show`.

#### AIX

`IToolBarFrameWindows` do not contain the minimize or maximize/restore buttons. To expand and collapse the toolbar contained in the toolbar frame window, you must call `IToolBar::expand` and `IToolBar::collapse`.

#### WIN

#### OS/2

`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 toolbar frame window to its owning frame window. This has the effect that the toolbar stays in the same location relative to its owner when the owner is moved.
- The right button allows you to expand and collapse the toolbar contained in the toolbar frame window. This allows you to shrink the toolbar to recover screen area when you are not actively using the toolbar and later expand the toolbar to show all toolbar controls.

#### WIN

The maximize/restore button on an `IToolBarFrameWindow` allows you to expand and collapse the toolbar contained in the toolbar frame window. This allows you to shrink the toolbar to recover screen area when you are not actively using the toolbar and later expand the toolbar to show all toolbar controls.

### Toolbar Buttons

The `IToolBarButton` class supports the following behavior:

- Toolbar buttons can accept text and a bitmap.
- You can cause bitmaps to be drawn on the toolbar buttons *transparently*. Drawing a bitmap transparently means changing some of the bits in your bitmap to match the color already on the display. The `IToolBarButton` class does this by selecting one color as the *transparent color*, then changing that transparent color to match the color on the display. By default, the transparent color is pink (255, 0, 255), but you may change it on a per button basis.
- You can create *latchable* toolbar buttons. Latchable buttons toggle between a latched (depressed) and unlatched (default) state.



### Standard Toolbar Buttons

The classes that comprise the toolbar are also shipped with common text, bitmaps, and default fly-over help strings for frequently used functions found on a toolbar. This ensures a common look among applications and products. In addition, toolbar buttons support a standard style to ensure a consistent look across applications.

### Direct Manipulation Support

You can use direct manipulation to rearrange the toolbar buttons within an application. To enable direct manipulation, either create the `IToolBar` object with the style `IToolBar::dragDrop`, or call `IToolBar::enableDragDrop`.

### Misfit Filtering

Toolbars may contain any object derived from `IWindow`, but generally, you would not use certain objects like notebooks and containers in a toolbar. Usually you would find the drop-down combination boxes and entry fields in a toolbar, but only in a horizontal toolbar. Because of their widths, these controls would waste too much space in a vertical toolbar. The `IToolBar` class provides *misfit filtering* that temporarily removes any windows in the toolbar that exceed a specified width, the *misfit width*, when you orient a toolbar vertically.

#### RELATED CONCEPTS

- “Chapter 23. Direct Manipulation” on page 313
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

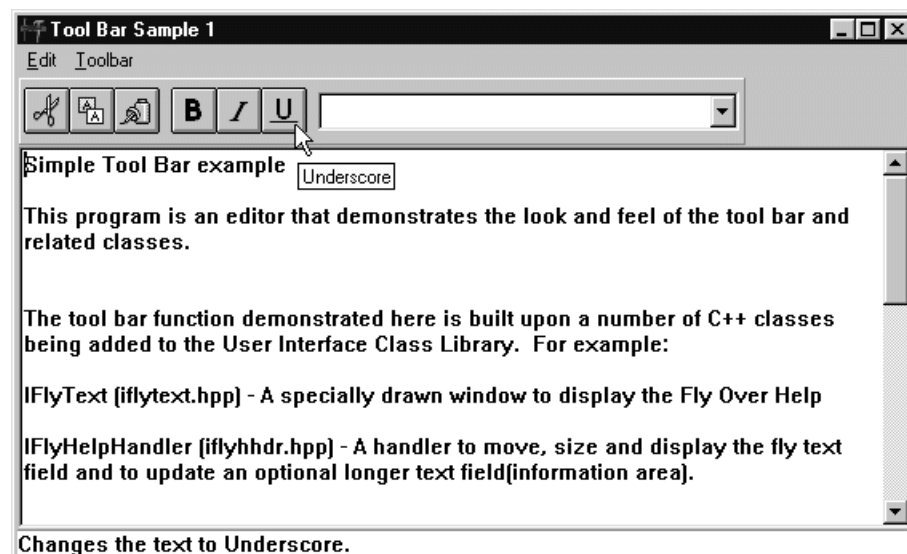
#### RELATED TASKS

- “Create a Toolbar”
- “Use Default Direct Manipulation” on page 316
- “Add Images to Drag Items” on page 331
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

---

## Create a Toolbar

The following figure, from the `Toolbar Sample 1`, shows an example of a toolbar.



This figure is created using the sample code found in the samples/ioc/tbar1 directory.

You can construct objects from the IToolBar class in the following ways:

1. The following IToolBar constructor creates a toolbar as the last toolbar in the toolbar area defined by the frameLocation. If groupWithPreceding is true, the toolbar is placed on the same row or column as the previous toolbar, if one exists. If an IToolBarContainer is needed for the location indicated, it is created when you use this constructor.

```
IToolBar (unsigned long identifier,
 IFrameWindow* owner,
 Location location = aboveClient,
 bool groupWithPreceding = false,
 const Style& style=defaultStyle());
```

2. Alternatively, you can construct a toolbar relative to an existing toolbar created with the first constructor (or with this constructor). This constructor adds the toolbar to the IToolBarContainer created when the precedingToolBar was created as follows:

```
IToolBar (unsigned long identifier,
 IToolBar* precedingToolBar,
 bool groupWithPreceding = false,
 const Style& style=defaultStyle());
```

### Customize Your Toolbar

When you create an object from the IToolBarButton class, you can use a standard format to ensure that all your buttons have the same common appearance. 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 toolbar buttons that have a style of IToolBarButton::standardFormat. The Open Class Library takes advantage of the standard formatting of toolbar buttons when painting the button. This therefore improves the performance.

The nested classes IToolBar::Style and IToolBarButton::Style provide a set of valid styles you can use upon construction for objects of the class.

#### RELATED CONCEPTS

“Chapter 16. Toolbars” on page 243  
“Chapter 23. Direct Manipulation” on page 313  
“Chapter 1. Windows” on page 1  
“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

Toolbar Sample 1  
“Create Help Information” on page 276  
“Use Default Direct Manipulation” on page 316  
“Add Images to Drag Items” on page 331  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

ICustomButton  
IFlyOverHelpHandler  
IFlyText

---

## Add a Toolbar

The following, from Toolbar Sample 2, shows a customized toolbar that implements fly-over help, direct manipulation support, and floating toolbars. This sample is found in the `samples/ioc/tbar2` directory.

The `tbar2.hpp` file does the following:

1. Defines the main window for the MLE, `EditorMLE`.
2. Defines three toolbars, `fileToolBar`, `editToolBar`, and `fontToolBar`.
3. Defines `IToolBarButton` objects, the buttons that this sample places on the toolbar.
4. Defines the toolbar settings notebook with the `ToolBarNotebook` class.

```
//-----
// Editor
//
// This class is the main window of the sample problem. It is
// responsible for creating and managing all of the windows that
// are used.
//-----
class Editor : public IFrameWindow
{
public:
 Editor();
 EditorMLE
 &editorWindow () { return editWindow; }
 IFont
 &editorFont () { return editFont; }
 Editor
 &updateFontToolBar ();
 IToolBar
 &toolbar (unsigned long id);
private:
 ITitle
 title;
 IToolBar
 fileToolBar,
 editToolBar,
 fontToolBar;
 IFlyText
 flyText;
 IStaticText
 infoText;
 IFlyOverHelpHandler
 flyHelpHandler;
 EditorMLE
 editWindow;
 EditorCommandHandler
 commandHandler;
 FontSelectHandler
 fontSelectHandler;
 IToolBarButton
 openButton,
 saveButton,
 cutButton,
 copyButton,
 pasteButton,
 boldButton,
 italicButton,
 underscoreButton;
 IComboBox
 fontCombo;
 IMenuBar
 menu;
```

```

IFont
 editFont;
};
//-----
// ToolBarNotebook
//-----
class ToolBarNotebook : public IFrameWindow
{
public:
 ToolBarNotebook (Editor& editor);
private:
 Editor
 &editorFrame;
 INotebook
 notebook;
};

```

The tbar2.cpp file does the following:

1. Constructs and show the editor (MLE) main window in the main function.
2. Constructs toolbars within the Editor constructor
3. Constructs the fly-over help short and long text and help handler within the Editor constructor
4. In the Editor constructor, adds the toolbar buttons to the toolbars with the addAsLast function
5. In the Editor constructor, adds the toolbar titles for floating toolbars with the setFloatingTitle function
6. In the Editor constructor, adds handlers for events, including fly-over help, to flyHelpHandler and commandHandler

```

/*****
* main - creates and shows the editor window
*****/
int main()
{
 Editor editor;
 editor.show();
 editor.setFocus();
 IApplication::current().run();
 return 0;
}
/*****
* Class Editor::Editor - Constructor for the editor window
*****/
Editor::Editor ()
 : IFrameWindow(ID_MAIN_WINDOW)
 , title(this)
 , fileToolBar(ID_FILE, this, IToolBar::aboveClient, false,
 IToolBar::classDefaultStyle
 | IToolBar::dragDrop)
 , editToolBar(ID_EDIT, &fileToolBar, true,
 IToolBar::classDefaultStyle
 | IToolBar::dragDrop)
 , fontToolBar(ID_FONT, &editToolBar, true,
 IToolBar::classDefaultStyle
 | IToolBar::dragDrop)
 , flyText(ID_FLYTEXT, this)
 , infoText(ID_INFOTEXT, this, this)
 , flyHelpHandler(&flyText, &infoText, 0, 0)
 , editWindow(ID_EDITOR, *this)
 , commandHandler(*this)
 , fontSelectHandler(*this)
 , openButton(IC_ID_OPEN, &fileToolBar, &fileToolBar,
 IRectangle(), IToolBarButton::defaultStyle() |
 IToolBarButton::dragDelete)

```

```

, saveButton(IC_ID_SAVE,&fileToolBar, &fileToolBar,
 IRectangle(), IToolBarButton::defaultStyle() |
 IToolBarButton::dragDelete)
, cutButton(IC_ID_CUT,&editToolBar, &editToolBar,
 IRectangle(), IToolBarButton::defaultStyle() |
 IToolBarButton::dragDelete)
, copyButton(IC_ID_COPY,&editToolBar, &editToolBar,
 IRectangle(), IToolBarButton::defaultStyle() |
 IToolBarButton::dragDelete)
, pasteButton(IC_ID_PASTE,&editToolBar, &editToolBar,
 IRectangle(), IToolBarButton::defaultStyle() |
 IToolBarButton::dragDelete)
, boldButton(IC_ID_BOLD, &fontToolBar, &fontToolBar,
 IRectangle(), IToolBarButton::defaultStyle() |
 IToolBarButton::transparentBitmap)
, italicButton(IC_ID_ITALIC, &fontToolBar, &fontToolBar,
 IRectangle(), IToolBarButton::defaultStyle() |
 IToolBarButton::transparentBitmap)
, underscoreButton(IC_ID_UNDERSCORE, &fontToolBar, &fontToolBar,
 IRectangle(), IToolBarButton::defaultStyle() |
 IToolBarButton::transparentBitmap)
, fontCombo(ID_FONTCOMBO, &fontToolBar, &fontToolBar, IRectangle(),
 IComboBox::classDefaultStyle & IComboBox::simpleType |
 IComboBox::readOnlyDropDownType)
, menu(ID_MAIN_WINDOW,this)
, editFont()
{
/*-----|
| Set the icon and the title
-----*/
 setIcon(id());
 title.setTitleText(ID_MAIN_WINDOW);
/*-----|
| Add buttons to the file toolbar
-----*/
 fileToolBar.addAsLast(&openButton,true)
 .addAsLast(&saveButton);
/*-----|
| Add buttons to the edit toolbar
-----*/
 editToolBar.addAsLast(&cutButton,true)
 .addAsLast(©Button)
 .addAsLast(&pasteButton);
/*-----|
| Add buttons to the font toolbar
-----*/
 fontToolBar.addAsLast(&boldButton,true)
 .addAsLast(&italicButton)
 .addAsLast(&underscoreButton)
 .addAsLast(&fontCombo,true);
/*-----|
| Set up the latchable style for font property buttons
| Note: When in bitmapAndTextView, latching a toolbar button will show the
| gray background for the bitmap. To eliminate the gray background, set
| the transparent color to gray and turn on the transparent style for the
| button. If you do not use the bitmapAndTextView and you have many buttons
| with the latchable style, consider not enabling the transparent style as
| transparency incurs additional overhead and increases performance time.
-----*/
 boldButton.enableLatching();
 italicButton.enableLatching();
 underscoreButton.enableLatching();
 boldButton.setTransparentColor(IColor::kPaleGray);
 italicButton.setTransparentColor(IColor::kPaleGray);
 underscoreButton.setTransparentColor(IColor::kPaleGray);
/*-----|
Load up front combo box with face names

```

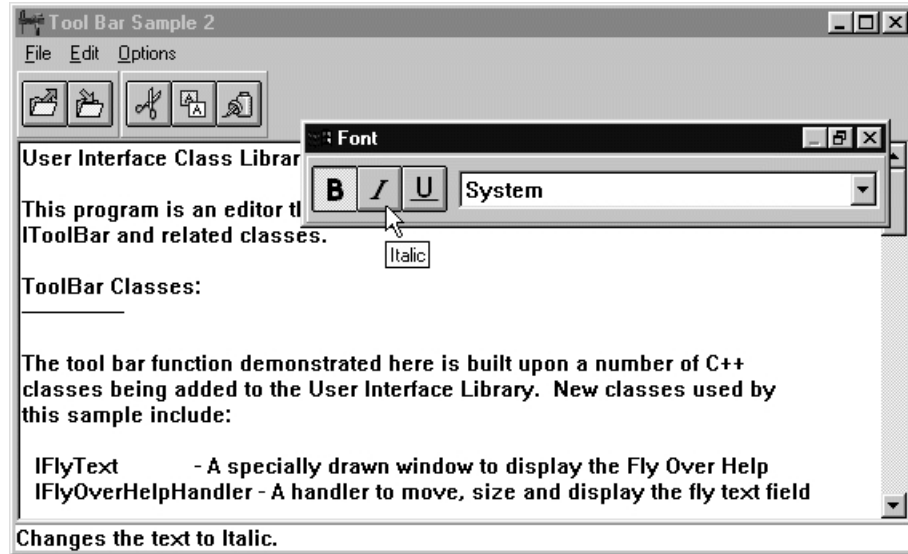
```

-----*/
fontCombo.setLimit(10);
#ifdef IC_MOTIF
IFont::FaceNameCursor fontCursor;
for (fontCursor.setToFirst(); fontCursor.isValid(); fontCursor.setToNext())
{
 IString faceName = IFont::faceNameAt(fontCursor);
 fontCombo.addAsLast(faceName);
 if (faceName.length() > fontCombo.limit())
 fontCombo.setLimit(faceName.length());
}
#endif
updateFontToolBar();
/*-----|
| Set up titles for toolbars when floating |
-----*/
fileToolBar.setFloatingTitle(ID_FILE);
editToolBar.setFloatingTitle(ID_EDIT);
fontToolBar.setFloatingTitle(ID_FONT);
/*-----|
| Setup the editor |
-----*/

setClient(&editWindow);
editWindow.setFont(editFont);
try
{
 editWindow.importFromFile("toolbar2.not");
}
catch(...)
{
 IMessageBox
 msgBox(this);
 msgBox.setTitle(STR_MSGBOX_TITLE);
 msgBox.show(STR_FILE_NOT_FOUND,
 IMessageBox::okButton |
 IMessageBox::warningIcon);
}
editWindow.setTop(1);
/*-----|
| Add the info frame extension |
-----*/
addExtension(&infoText, IFrameWindow::belowClient);
/*-----|
| Set up and add the help handler |
-----*/
flyHelpHandler.setLongStringTableOffset(OFFSET_INFOTEXT);
flyHelpHandler.setDefaultText("\0");
flyHelpHandler.handleEventsFor(&fileToolBar);
flyHelpHandler.handleEventsFor(&editToolBar);
flyHelpHandler.handleEventsFor(&fontToolBar);
/*-----|
| Attach the Command Handler to frame and toolbar |
-----*/
commandHandler.handleEventsFor(this);
commandHandler.handleEventsFor(&fileToolBar);
commandHandler.handleEventsFor(&editToolBar);
commandHandler.handleEventsFor(&fontToolBar);
/*-----|
| Add the handler to change the font |
-----*/
fontSelectHandler.handleEventsFor(&fontCombo);
moveSizeToClient(IRectangle(IPoint(100,100),
 ISize(editFont.avgCharWidth()*80,
 editFont.maxCharHeight()*15)));
}

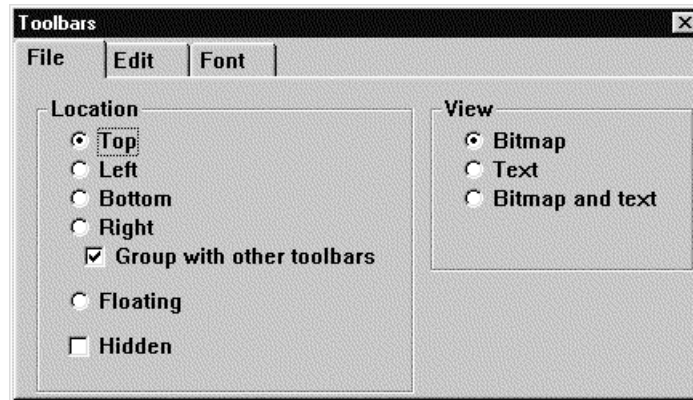
```

The resulting MLE and toolbars are displayed below.



You can use the settings notebook to choose where the toolbar appears and what appears on each toolbar button. In the preceding figure, there are three toolbars with different toolbar buttons. The floating toolbar, containing the font information, has been pinned to the owner window. The edit toolbar has been placed on the bottom of the window and contains both text and bitmaps on the buttons.

The settings notebook for the toolbars is displayed below.



You can find this sample in the samples/ioc/tbar2 directory.

#### RELATED CONCEPTS

“Chapter 16. Toolbars” on page 243

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

Toolbar Sample 2

“Create a Toolbar” on page 245

“Create Toolbar Buttons” on page 252

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

IToolBar

---

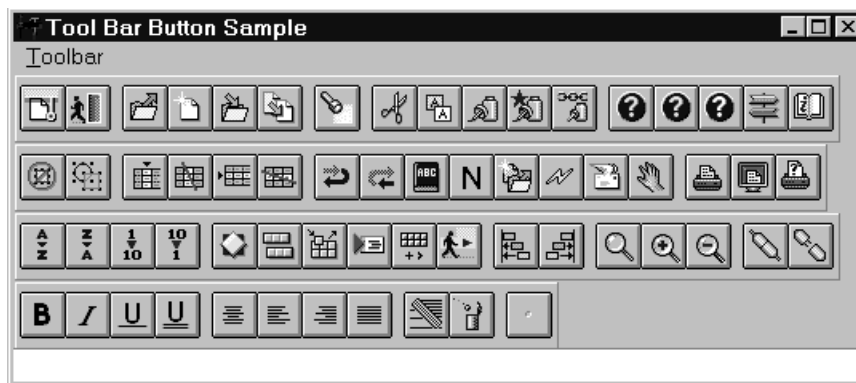
## Create Toolbar Buttons

For creating toolbar buttons, the Open Class Library supplies a set of bitmaps and associated bitmap and help text. These bitmaps are based on the IBM/Lotus approved set available for user interface application development.

The IBM/Lotus toolbar graphics consist of approximately 60 graphical images, each representing a common action. This collection was comprised from graphical images used in a number of IBM and Lotus® applications.

The Open Class Library supplies these images (in the form of bitmaps), plus a few others for your use in toolbar buttons. In addition to the button bitmaps, the Open Class Library also supplies button text and short and long help text. The button text is used when toolbar buttons are set to text or to text and bitmap view. The help text is used to provide additional information to the user about the function of the button. The code samples display the short text for each button in fly-over help (sometimes called hover help), when the mouse is moved over a button. The samples display the long text for each button in an information area at the bottom of the main window. This text is also triggered by the movement of the mouse over a toolbar button.

The resource IDs for these common bitmaps are defined in the `icconst.h` file. The following figure shows the commonly used bitmaps shipped with the IBM Open Class Library. The figure was generated from the Toolbar Button Sample. You can find this sample in the `samples/ioc/tbarbutn` directory:



When you build applications that use toolbars or any other resources provided, you must supply the Open Class Library resource DLL with your application. See the topic [Package and Distribute an IBM Open Class Application](#) for more information.

The following is a list of the new toolbar button bitmaps and their associated text. The identifier (ID) is used when creating the toolbar button.



Note: The \n in the button text of certain buttons indicates a line break. When toolbar button text is displayed, the \n is removed, and the remaining text is continued on the next line:

**ID:** IC\_ID\_CANCELOPERATION  
**Button Text:** Cancel  
**Short Help Text:** Cancel Current Operation  
**Long Help Text:** End the current operation and discard any changes  
**ID:** IC\_ID\_EXIT  
**Button Text:** Exit  
**Short Help Text:** Exit  
**Long Help Text:** Leave the application  
**ID:** IC\_ID\_OPEN  
**Button Text:** Open  
**Short Help Text:** Open  
**Long Help Text:** Open another document/object  
**ID:** IC\_ID\_NEW  
**Button Text:** New  
**Short Help Text:** New  
**Long Help Text:** Create a new document/object and open it  
**ID:** IC\_ID\_SAVE  
**Button Text:** Save  
**Short Help Text:** Save  
**Long Help Text:** Save your current work  
**ID:** IC\_ID\_SAVE\_AS  
**Button Text:** Save As  
**Short Help Text:** Save As  
**Long Help Text:** Save a copy of the current document/object with a new name  
**ID:** IC\_ID\_LOCATE  
**Button Text:** Find...  
**Short Help Text:** Find  
**Long Help Text:** Find within the current document/object  
**ID:** IC\_ID\_CUT  
**Button Text:** Cut  
**Short Help Text:** Cut  
**Long Help Text:** Cut the selected object to the Clipboard  
**ID:** IC\_ID\_COPY  
**Button Text:** Copy  
**Short Help Text:** Copy  
**Long Help Text:** Copy the selected object to the Clipboard  
**ID:** IC\_ID\_PASTE  
**Button Text:** Paste  
**Short Help Text:** Paste  
**Long Help Text:** Paste from the Clipboard  
**ID:** IC\_ID\_PASTESPECIAL  
**Button Text:** Paste\nSpecial...  
**Short Help Text:** Paste Special  
**Long Help Text:** Paste, link, or embed from the Clipboard  
**ID:** IC\_ID\_PASTELINK  
**Button Text:** Paste Link  
**Short Help Text:** Paste Link  
**Long Help Text:** Paste a link from the Clipboard  
**ID:** IC\_ID\_HELP  
**Button Text:** Help  
**Short Help Text:** Help  
**Long Help Text:** Display help text  
**ID:** IC\_ID\_GENERALHELP  
**Button Text:** General\nHelp  
**Short Help Text:** General Help  
**Long Help Text:** Display general help information  
**ID:** IC\_ID\_HELPINDEX  
**Button Text:** Help Index  
**Short Help Text:** Help Index  
**Long Help Text:** Display an index of all the available help  
**ID:** IC\_ID\_HELP\_TUTORIAL  
**Button Text:** Tutorial  
**Short Help Text:** Take a Tutorial

**Long Help Text:** Take a tutorial explaining how to use the application  
**ID:** IC\_ID\_USINGHELP  
**Button Text:** Using Help  
**Short Help Text:** Using Help  
**Long Help Text:** Learn how to use help  
**ID:** IC\_ID\_DESELECTALL  
**Button Text:** Deselect\nAll  
**Short Help Text:** Deselect All  
**Long Help Text:** Deselect all objects  
**ID:** IC\_ID\_SELECTALL  
**Button Text:** Select All  
**Short Help Text:** Select All  
**Long Help Text:** Select all objects  
**ID:** IC\_ID\_INSERTCOLUMN  
**Button Text:** Insert\nColumn  
**Short Help Text:** Insert Column  
**Long Help Text:** Insert a column into the table  
**ID:** IC\_ID\_DELETECOLUMNS  
**Button Text:** Delete\nColumn  
**Short Help Text:** Delete Column  
**Long Help Text:** Delete the selected column(s)  
**ID:** IC\_ID\_INSERTROW  
**Button Text:** Insert Row  
**Short Help Text:** Insert Row  
**Long Help Text:** Insert a row into the table  
**ID:** IC\_ID\_DELETEROWS  
**Button Text:** Delete Row  
**Short Help Text:** Delete Row  
**Long Help Text:** Delete the selected row(s)  
**ID:** IC\_ID\_UNDO  
**Button Text:** Undo  
**Short Help Text:** Undo  
**Long Help Text:** Undo the last action you took  
**ID:** IC\_ID\_REDO  
**Button Text:** Redo  
**Short Help Text:** Redo  
**Long Help Text:** Redo the action you just undid  
**ID:** IC\_ID\_CHECKSPELLING  
**Button Text:** Spelling  
**Short Help Text:** Check Spelling  
**Long Help Text:** Check the spelling  
**ID:** IC\_ID\_NORMAL  
**Button Text:** Normal\nText  
**Short Help Text:** Normal Text  
**Long Help Text:** Remove emphasis from the selected text  
**ID:** IC\_ID\_ABOUT  
**Button Text:** Product\nInfo  
**Short Help Text:** Product Info  
**Long Help Text:** Version, copyright and legal information  
**ID:** IC\_ID\_UPDATE  
**Button Text:** Update  
**Short Help Text:** Update  
**Long Help Text:** Update the current view  
**ID:** IC\_ID\_MAILSEND  
**Button Text:** Send...  
**Short Help Text:** Send This to Someone  
**Long Help Text:** Send the current document/object to another person  
**ID:** IC\_ID\_MOVEPAGES  
**Button Text:** Move Page  
**Short Help Text:** Move Page  
**Long Help Text:** Move the current page to another position  
**ID:** IC\_ID\_PRINT  
**Button Text:** Print...  
**Short Help Text:** Print  
**Long Help Text:** Print the current document/object  
**ID:** IC\_ID\_PRINTPREVIEW  
**Button Text:** Print\nPreview

**Short Help Text:** Print Preview  
**Long Help Text:** See how the document looks when printed  
**ID:** IC\_ID\_PRINTSETUP  
**Button Text:** Print\nSetup  
**Short Help Text:** Print Setup  
**Long Help Text:** Display the current settings for printing  
**ID:** IC\_ID\_SORTAZ  
**Button Text:** Sort A-Z  
**Short Help Text:** Sort Ascending (alphabetical)  
**Long Help Text:** Sort the selected objects in ascending alphabetical order  
**ID:** IC\_ID\_SORTZA  
**Button Text:** Sort Z-A  
**Short Help Text:** Sort Descending (alphabetical)  
**Long Help Text:** Sort the selected objects in descending alphabetical order  
**ID:** IC\_ID\_SORTUP  
**Button Text:** Sort 1-9  
**Short Help Text:** Sort Ascending (numeric)  
**Long Help Text:** Sort the selected objects in ascending numerical order  
**ID:** IC\_ID\_SORTDOWN  
**Button Text:** Sort 9-1  
**Short Help Text:** Sort Descending (numeric)  
**Long Help Text:** Sort the selected objects in descending numerical order  
**ID:** IC\_ID\_SETTINGS  
**Button Text:** Properties  
**Short Help Text:** Properties  
**Long Help Text:** Show properties of the current document/object  
**ID:** IC\_ID\_TOOLBAR  
**Button Text:** Show\nToolbar  
**Short Help Text:** Show Toolbar  
**Long Help Text:** Display the toolbar for the application  
**ID:** IC\_ID\_TOOLBARSETUP  
**Button Text:** Toolbar\nSetup  
**Short Help Text:** Toolbar Setup  
**Long Help Text:** Customize the setup of your toolbar  
**ID:** IC\_ID\_RUNSCRIPT  
**Button Text:** Run\nScript...  
**Short Help Text:** Run a Script  
**Long Help Text:** Run a script  
**ID:** IC\_ID\_VPAPALETTE  
**Button Text:** Show\nActions...  
**Short Help Text:** Show Actions and Properties You Can Add  
**Long Help Text:** Show a palette of available actions and properties you can add to this application  
**ID:** IC\_ID\_TOUR  
**Button Text:** Tour...  
**Short Help Text:** Take a Tour  
**Long Help Text:** Take a tour of the application's features and learn how to use them  
**ID:** IC\_ID\_LEFTALIGNOBJ  
**Button Text:** Left Align  
**Short Help Text:** Left Align Objects  
**Long Help Text:** Left align the selected objects  
**ID:** IC\_ID\_RIGHTALIGNOBJ  
**Button Text:** Right Align  
**Short Help Text:** Right Align Objects  
**Long Help Text:** Right align the selected objects  
**ID:** IC\_ID\_ZOOM  
**Button Text:** Zoom  
**Short Help Text:** Zoom  
**Long Help Text:** Zoom selected area , so it fills the work area  
**ID:** IC\_ID\_ZOOMIN  
**Button Text:** Zoom In  
**Short Help Text:** Zoom In  
**Long Help Text:** Zoom in closer to the current document/object  
**ID:** IC\_ID\_ZOOMOUT  
**Button Text:** Zoom Out  
**Short Help Text:** Zoom Out  
**Long Help Text:** Zoom out farther from the current document/object  
**ID:** IC\_ID\_LOGON

**Button Text:** Logon  
**Short Help Text:** Logon  
**Long Help Text:** Log on to the system  
**ID:** IC\_ID\_LOGOFF  
**Button Text:** Logoff  
**Short Help Text:** Logoff  
**Long Help Text:** Log off the system  
**ID:** IC\_ID\_BOLD  
**Button Text:** Bold  
**Short Help Text:** Bold  
**Long Help Text:** Make the selected text bold  
**ID:** IC\_ID\_ITALIC  
**Button Text:** Italic  
**Short Help Text:** Italic  
**Long Help Text:** Make the selected text Italic  
**ID:** IC\_ID\_UNDERSCORE  
**Button Text:** Underline  
**Short Help Text:** Underline  
**Long Help Text:** Underline the selected text  
**ID:** IC\_ID\_DOUBLEUNDERSCORE  
**Button Text:** Double Underline  
**Short Help Text:** Double Underline  
**Long Help Text:** Double underline the selected text  
**ID:** IC\_ID\_CENTER  
**Button Text:** Align Center  
**Short Help Text:** Align Center  
**Long Help Text:** Center align the selected text  
**ID:** IC\_ID\_LEFT  
**Button Text:** Align Left  
**Short Help Text:** Align Left  
**Long Help Text:** Left align the select text  
**ID:** IC\_ID\_RIGHT  
**Button Text:** Align Right  
**Short Help Text:** Align Right  
**Long Help Text:** Right align the selected text  
**ID:** IC\_ID\_JUSTIFY  
**Button Text:** Justify  
**Short Help Text:** Full Justify  
**Long Help Text:** Fully justify the selected text  
**ID:** IC\_ID\_CLEAR  
**Button Text:** Clear  
**Short Help Text:** Clear  
**Long Help Text:** Clear the selected object(s)  
**ID:** IC\_ID\_DELETE  
**Button Text:** Delete  
**Short Help Text:** Delete  
**Long Help Text:** Delete the selected object(s)

The new Open Class toolbar bitmaps differ in some ways from the previous Open Class toolbar bitmaps. In addition to the change in the number of bitmaps, their size, default transparent state, and the number of lines needed to display text have changed.

The new bitmaps consist of the 61 listed above. There are also 13 available in the current product. Each of these is also included in the new group. The actual graphic depiction of these 13 is replaced with a new representation. The ID you use to refer to the current bitmaps references the new bitmaps.

You can still reference the current bitmaps by using a new ID that corresponds to these bitmaps. These new IDs are as follows:

- IC\_ID\_TRANS\_NEW
- IC\_ID\_TRANS\_ABOUT
- IC\_ID\_TRANS\_OPEN

- IC\_ID\_TRANS\_SAVE
- IC\_ID\_TRANS\_PRINT
- IC\_ID\_TRANS\_LOCATE
- IC\_ID\_TRANS\_CUT
- IC\_ID\_TRANS\_COPY
- IC\_ID\_TRANS\_PASTE
- IC\_ID\_TRANS\_HELP
- IC\_ID\_TRANS\_BOLD
- IC\_ID\_TRANS\_ITALIC
- IC\_ID\_TRANS\_UNDERSCORE
- IC\_ID\_TRANS\_SETTINGS
- IC\_ID\_TRANS\_COPYTO

Each of the new bitmaps has a non-transparent gray background. This means that the solid background area of a bitmap does not show through the bitmap. Changing the default state of buttons from transparent to non-transparent is being done for performance. Displaying a bitmap with transparency involves creating a mask bitmap, which must be combined with the original bitmap and drawn to the screen. This operation is quite a bit slower than drawing a single non-transparent bitmap to the screen. In a toolbar with 10 or more buttons, this performance savings is clearly visible.

The background gray color selection is being carefully chosen. On Windows, the background gray has a Red, Green, Blue (RGB) value of (192, 192, 192). This matches the default gray used by Win32. On OS/2, the background RGB values are (204, 204, 204), to match the default gray used by OS/2 4.0. On AIX, the background gray color is (204, 204, 204).

To provide support for transparent buttons, we introduced a new style called `IToolBarButton::transparentBitmap`. If you want transparent behaviour for a particular button, you need to specify this style on button creation. In addition to specifying this style, you also need to define a color to be used for transparency. By default, a transparent color is initially defined. This color, pink, has an RGB value of (255,0,255). The following example sets up transparent behavior for one of the new button bitmaps:

```
IToolBarButton * cutButton = new IToolBarButton(
 IC_ID_CUT, &toolbar, &toolbar,
 IRectangle(), IToolBarButton::defaultStyle() |
 IToolBarButton::transparentBitmap);
//By default, a transparent color of RGB value (255,0,255) has
//already been set. Reset the color to match the background in the bitmap.
IColor *transparentColor = new IColor(192,192,192); //Win32
cutButton->setTransparentColor(*transparentColor);
```

Each of the new bitmaps is slightly larger. This in turn makes the toolbar button slightly larger. This change is being made for the following reasons: The larger size makes the buttons more readable. There is more room to depict each graphical image. The buttons are symmetrically square rather than rectangular, which with their larger size, makes them more consistent with the Lotus Smart Suite applications and other Win32 applications.

The button sizes are as follows:

|                        | Current Bitmaps | New Bitmaps |
|------------------------|-----------------|-------------|
| Bitmap size:           | 20x17           | 22x22       |
| Resulting button size: | 27x24           | 29x29       |

To differentiate between similar images and actions, the button text (which can be seen when the button is displayed in text or bitmap and text view) is two lines on some buttons. These are:

- IC\_ID\_PASTESPECIAL, with text of: "Paste\nSpecial..."
- IC\_ID\_GENERALHELP, with text of: "General\nHelp"
- IC\_ID\_DESELECTALL, with text of: "Deselect\nAll"
- IC\_ID\_INSERTCOLUMN, with text of: "Insert\nColumn"
- IC\_ID\_DELETECOLUMNS, with text of: "Delete\nColumn"
- IC\_ID\_NORMAL, with text of: "Normal\nText"
- IC\_ID\_ABOUT, with text of: "Product\nInfo"
- IC\_ID\_PRINTPREVIEW, with text of: "Print\nPreview"
- IC\_ID\_PRINTSETUP, with text of: "Print\nSetup"
- IC\_ID\_TOOLBAR, with text of: "Show\nToolbar"
- IC\_ID\_TOOLBARSETUP, with text of: "Toolbar\nSetup"
- IC\_ID\_RUNSCRIPT, with text of: "Run\nScript..."
- IC\_ID\_VPAPALETTE, with text of: "Show\nActions..."
- IC\_ID\_DOUBLEUNDERSCORE, with text of: "Double\nUnderline"
- IC\_ID\_CENTER, with text of: "Align\nCenter"
- IC\_ID\_LEFT, with text of: "Align\nLeft"
- IC\_ID\_RIGHT, with text of: "Align\nRight"

The default for standard buttons (those created with the `IToolBarButton::standardFormat` style) is still to display only one line of text on a button when the button is displayed in text or bitmap and text view. This is being left unchanged to maintain consistency with existing Open Class toolbar applications and because the majority of the buttons have only one line of text.

If you are using one of the pre-defined buttons above, and your toolbar buttons are standard format, you can set the default number of lines of text to be displayed for toolbar buttons to 2. You call `IToolBarButton::setStandardTextLines`, as follows:

```
IToolBarButton::setStandardTextLines(2);
```

If you do not set the standard text lines to 2, the second line of text is not displayed.

The rationale behind adding a second line to display the extra text, is so that the toolbar lays-out better in an application. If the extra text were to be displayed on one long line, the buttons would need to be wide rectangular shaped to fit the text.

Each of the new toolbar bitmaps contains associated short and long help text. The help text is used to provide additional information to the user about the function of the button. Most often, application developers choose to display short text for a button in fly over help, which appears when the mouse is moved over a button. Developers often choose to display long text for a button in an information area at the bottom of a window. This text is also triggered by the movement of the mouse over a toolbar button.

You can still supply your own short and long help text, as you do in previous versions of Open Class. However, if you want to use the Open Class supplied help text, you can do so in only a few lines of code:

```
//Tell the IFlyOverHelpHandler to search for text in the Open Class
//supplied resource library.
flyHelpHandler.setResourceLibrary(
 IApplication::current().resourceLibrary().fileName());
//Tell the IFlyOverHelpHandler the offsets for the help text.
flyHelpHandler.setFlyTextStringTableOffset(IC_FLYTEXTOFFSET);
flyHelpHandler.setLongStringTableOffset(IC_LONGTEXTOFFSET);
```

If you want to supply your own help text, first ensure that the IFlyOverHelpHandler's resource library is the one bound to your application (which is the default for IFlyOverHandler). Second, associate your text string with the button ID in your resource file. You can optionally define an ID offset, as is done above.

If you have an existing Open Class application, you get the new bitmaps automatically with no changes needed to your application. Of course, if you want to take advantage of the short and long help text, or use one of the added bitmaps that have two lines of button text, you need to make some modifications to your program.

To revert back to the current bitmaps, you need to change your application to use the new IDs that refer to these bitmaps. These IDs are listed above. Secondly, you need to create each button with the IToolBarButton::transparentBitmap style.

If the window ID is in the Open Class library reserved range, the bitmap and text are loaded from the Open Class Library's resource library. Otherwise, the bitmap and text are loaded from the default user defined resource library. If no bitmap is found in the appropriate library, a default bitmap is loaded.

#### **RELATED CONCEPTS**

"Chapter 16. Toolbars" on page 243

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

#### **RELATED TASKS**

Toolbar Button Sample

"Create a Toolbar" on page 245

"Add a Toolbar" on page 247

Package and Distribute an IBM Open Class Application

"Write an Event Handler" on page 25

"Task and Samples Cross-Reference Table" on page 445





---

## Chapter 17. Common Dialogs

Common dialog boxes make it easier for you to develop applications. These dialog boxes provide common, consistent functions that applications can use to help the end user increase his productivity. Although a wide range of function is available, the simplest applications can make use of these to provide a consistent look-and-feel with a minimal coding effort.

The Open Class Library provides several classes to enable you to use common dialog boxes such as the **Open Filename** dialog, the **Save As Filename** dialog, and the **Font Selection** dialog.

A *dialog box* is a special, short-lived window that you use to display information and receive input from the user in a structured dialog format. The information is typically related to a particular action being performed by the application.

If you develop portable applications, use a canvas for dialog windows.

### RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

### RELATED TASKS

“Specify File Dialog Information”

“Create a File Dialog” on page 263

“Specify Font Dialog Information” on page 266

“Hello World Version 4: Adding Dialogs and Push Buttons” on page 420

“Create a Split Canvas” on page 165

“Specify Font Dialog Information” on page 266

“Create a Font Dialog” on page 267

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Specify File Dialog Information

Many applications work with information stored in files. Whenever a user wants to retrieve information stored in a file, the user must select the name of the file from which to get the data. The same holds true when the user wants to save any work that is in progress for future reference.

The *file dialog* enables a user to specify a file to be opened or a file name in which current work is to be saved. It provides basic functions that you can extend to meet the requirements of your application. These basic functions give users the ability to:

- Display and select from a list of drives, directories, and files.
- Enter a file name directly.
- Filter the file names before they are displayed.
- Display active network connections.
- Interact with a single-selection or multiple-selection file dialog.
- Interact with a modal or modeless file dialog.

The Open Class Library provides the `IFileDialog` and `IFileDialog::Settings` classes to create and display file dialogs in your application.

`IFileDialog::Settings` is a *settings* type class, which defines data, styles, and attributes of a file dialog. You create `IFileDialog::Settings` objects and use them to create an `IFileDialog` object. This object governs the usage and appearance of the file dialog, as well as returning the file name choice (or choices in the case of multiple selection) that the user made.

Before you can create an `IFileDialog`, you first need to create an `IFileDialog::Settings` object. To create this settings object, follow these steps.

1. Create a default `IFileDialog::Settings` object, as shown in the following example:

```
IFileDialog::Settings fsettings;
```

2. Initialize this settings object using the member functions available. While the default settings may be suitable for many applications, most application developers modify one or more of these settings. Some of the more common settings that you can change are the following:

#### **Dialog type**

The `setOpenDialog` and `setSaveAsDialog` member functions are used to specify the type of file dialog that will be displayed. The default type of the file dialog is an Open dialog. If you want a **Save As** dialog, use the `setSaveAsDialog` member function.

#### **Initial file name displayed**

The `setFileName` member function is used to specify the initial file name that is shown when the file dialog is displayed.

#### **Dialog position**

The `setPosition` member function is used to specify the placement of the file dialog, relative to the window that owns the dialog.

#### **Dialog title**

The `setTitle` member function is used to specify the text that is displayed in the title bar of the dialog.

The following example shows some of these being initialized for the object created in the previous example:

```
fsettings.setSaveAsDialog(); // The dialog type is Save As,
fsettings.setFileName("example.txt"); // the initial filename displayed is example.txt,
fsettings.setTitle(STR_MYAPP); // and the title is loaded from a resource file
```

Now that you have a settings object created, you can create an `IFileDialog` object. There are several ways to construct an `IFileDialog` object, but in general you need to provide the owner and parent windows and a settings object. Optionally, you can provide different dialog styles to control the appearance and functionality of the dialog, but these have defaults and do not need to be modified for many applications. The `IFileDialog` object can be constructed using the following statements:

```
IFileDialog* fdialog = new IFileDialog(// Create file open dialog
 desktopWindow(), // parent is desktop
 this, // owner is me
 fsettings); // use the settings defined above.
```

When the dialog is dismissed, you need to check the actions of the end user to determine if he has selected any file names to be used by your application. You can test the response from the file dialog using the `pressedOK` member function.

This member function returns true if the user ended the dialog by pressing the **OK** push button. It returns false if the user pressed **Cancel** or an error occurred. The test can be done with the following statement:

```
if (fdialog.pressedOK())
```

If the end user pressed **OK**, then the file name (or names) selected can be retrieved from the dialog using the `fileName` member function. This function returns the fully qualified file name, which includes drive and directory information. For a single selection style of the file dialog, the following statement is used:

```
IString fname = fdialog.fileName();
```

For a multiple selection style of the file dialog, the following statements are used:

```
int count = fdialog.selectedFileCount(); // Retrieve # of files selected
int i;
for (i=1; i <= count; i++) // For each file selected,
{ // retrieve the fully-qualified
 IString fname = fdialog.fileName(i); // file name into a string
 // ... // Process it
} // and then get the next one
```

For a complete code sample, see the `openFile` member function in the `ahellow6.cpp` file of Hello World Version 6.

#### RELATED TASKS

“Create a File Dialog”

“Chapter 17. Common Dialogs” on page 261

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED REFERENCES

“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443

`IFileDialog`

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Create a File Dialog

The following code, from the multiline edit sample, shows you how to create both an **Open** file dialog and a **Save As** file dialog. You can find the Multiline Edit Sample in the `samples/ioc/mle` directory:

1. In the constructor of your main window, declare functions that will display your file dialogs. In the MLE sample, the `openFile()` function will create an **Open** file dialog, and the `saveAsFile()` function will create a **Save As** file dialog:

```
class AEditorWindow : public IFrameWindow
 , public ICommandHandler
 , public IMenuHandler
 , public AUserMessageHandler
{
public:
 AEditorWindow(unsigned long windowId);
 AEditorWindow
 &openFile(bool fUseThread),
 &saveFile(),
 &saveAsFile(),
 &openFont(),
```

```

 &loadOnThread(unsigned long eventId),
 &load());
 // ...
};

```

2. Define the function that will create an **Open** file dialog. The MLE sample first creates an `IFileDialog::Settings` object called `fdSettings`. This object lets you specify the following characteristics of an `IFileDialog` object:

- 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

The MLE sample uses the `setOpenDialog()` function to specify that the file dialog will be an **Open** file dialog. The sample uses the `setFileName()` function to specify the default file name selection.

The `IFileDialog` constructor takes the `fdSettings` object as one of its arguments:

```

AEditorWindow& AEditorWindow::openFile(bool fUseThread)
{
 IFileDialog::Settings fdSettings;
 fdSettings.setOpenDialog();
 if (filename.size())
 fdSettings.setFileName(filename);
 else
 fdSettings.setFileName(DEFAULT_FILE_SPEC);
 IFileDialog fileDlg(desktopWindow(), this, fdSettings);
 if (fileDlg.pressedOK())
 {
 filename=fileDlg.fileName();
 titleBar.setObjectText(IResourceId(WND_MAIN));
 titleBar.setViewText(filename);
 if (filename.size())
 {
 // determine whether to use thread
 // Since GUI operations are not thread
 // safe on Motif, do not use threads for
 // loading files into the MLE.

 if (fUseThread)
 {
 // disable file menu items to avoid them
 // being reselected while thead running
 setFileMenuitemsState(false);
 // create our thread function and
 // specify the message id we want
 // posted back
 AThreadFn *atmFn = new AThreadFn(*this, UWM_THREADEND);
 // dispatch thread to run function
 // thread will have a PM environment
 IThread thread(atmFn, IThread::defaultAutoInitGUI());
 }
 else
 {
 load();
 }
 }
 }
 return *this;
}

```

3. Define the function that will create a **Save As** file dialog. The MLE does this in a similar manner to the `openFile()` function defined above. It uses the `setSaveAsDialog()` function to specify that the file dialog will be a **Save As** file dialog:

```
AEditorWindow& AEditorWindow::saveAsFile()
{
 IFileDialog::Settings fdSettings;
 fdSettings.setSaveAsDialog();
 fdSettings.setFileName(filename);
 IFileDialog fileDlg(desktopWindow(), this, fdSettings);
 if (fileDlg.pressedOK())
 {
 filename=fileDlg.fileName();
 if (filename.size())
 {
 mle.exportToFile(filename);
 titleBar.setObjectText(IResourceId(WND_MAIN));
 titleBar.setViewText(filename);
 }
 }
 return *this;
}
```

The following figure shows the **Open** file dialog created using the preceding example:



You can find the Multiline Edit Sample in the `samples/ioc/mle` directory.

#### RELATED CONCEPTS

- “Chapter 17. Common Dialogs” on page 261
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- Multiline Edit Sample
- “Specify File Dialog Information” on page 261
- “Specify Font Dialog Information” on page 266
- “Create a Font Dialog” on page 267
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

---

## Specify Font Dialog Information

Many applications display information to users in a variety of ways, many of them textually based. A common action that users perform is selecting a font with which to display or print this textual information. A font represents all of the characters of one style of type in one size.

The *font dialog* provides a dialog you can implement to enable users to view and select the font family names, styles, and sizes available in an application. It provides basic functions that you can extend to meet the requirements of your application. These basic functions give users the ability to display and select from a list of the following:

- Font family names installed on the system
- Styles for each font
- The available sizes for each font
- The emphasis styles available for each font.

In addition, users can view their selections using a sample character string in a preview area, and interact with a modal or modeless font dialog.

The Open Class Library provides the `IFontDialog` class to create and display font dialogs in your application.

`IFontDialog::Settings` is a settings type class that defines data, styles and attributes of a font dialog. You create `IFontDialog::Settings` objects and use them to create an `IFontDialog` object. This object governs the usage and appearance of the font dialog, as well as returning the font chosen by the user.

Before you can create an `IFontDialog`, you first need to create an `IFontDialog::Settings` object. To create this settings object, follow these steps:

1. Create an `IFontDialog::Settings` object, passing it the initial font to be used, as shown in the following example:

```
IFont tempFont(&hello); // Create a font from the hello window
IFontDialog::Settings // Create the initial settings object
fsettings(&tempFont); // from the font
```

2. Initialize this settings object using the many member functions available. While the default settings are suitable for many applications, most application developers modify one or more of these settings. Some of the more common settings that you can change are the following:

### Preview text

The `setPreviewText` member function is used to specify the text string that is used to demonstrate how the selected font will look if selected.

### Dialog position

The `setPosition` member function is used to specify the placement of the font dialog, relative to the window that owns the dialog.

### Dialog title

The `setTitle` member function is used to specify the text that is displayed in the title bar of the dialog.

The following example shows these being initialized for the object created in the previous example:

```
fsettings.setPreviewString("ABCDabcd"); // Use this string to demo type,
fsettings.setPosition(IPoint(50, 20)); // position at 50,20 in owner,
fsettings.setTitle("Choose font"); // and the title is Choose font
```

Now that you have a settings object created, you can create an `IFontDialog` object. There are several ways you can construct an `IFontDialog` but in general, you need to provide the owner and parent windows and a settings object. Optionally, you can provide different dialog styles to control the appearance and functionality of the dialog, but these have defaults and do not need to be modified for many applications. The `IFontDialog` object can be constructed using the following statements:

```
IFontDialog* fdialog = new IFontDialog(// Create font dialog
 desktopWindow(), // Parent is desktop
 this, // Owner is me
 fsettings); // Use the settings defined above
```

When the dialog is dismissed, you need to check the actions of the end user to determine if he has selected a valid font to be used by your application. You can test the response from the font dialog using the `pressedOK` member function. This member function returns true if the user ended the dialog by pressing the **OK** push button. It returns false if the user pressed **Cancel** or an error occurred. The test can be done with the following statement:

```
if (fdialog.pressedOK())
```

If the end user did press **OK**, the `tempFont` object that was used to initialize the settings object is automatically updated to be an `IFont` object that reflects the users selection. Your application can now use this font to change how you display textual information to your users. This is done using either the `setWindowFont` member function (a function of `IFont` which accepts a window to apply the font to) or the `setFont` member function (a function of `IWindow` which accepts a font to apply to the window). can be used. The latter is shown in the following statement:

```
hello.setFont(&tempFont);
```

#### RELATED CONCEPTS

“Chapter 18. Fonts” on page 271

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Create a Font Dialog”

“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443

“Chapter 17. Common Dialogs” on page 261

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

`IStaticText`

---

## Create a Font Dialog

The following section describes how to create a font dialog using the Hello World version 6 application. The following code comes from the `ahello6.cpp` file:

```
AHelloWindow &
 AHelloWindow :: setHelloFont()
{
 /*-----|
 | Set the text in the information area from the font dialog information |
 | string in the resource file. |
```

```

Create a font from the current font of the hello static text object.
Create a font settings object from the tempFont object.
Set the title to be used in the font dialog from a string resource in
the resource file.
Start the font dialog by creating an IFontDialog object. The font
dialog is created as a modal dialog window with the desktopWindow
as the parent, the AHelloWindow frame as the owner, resetButton
style which supplies the user with a way to reset the fields in the
dialog, and the font settings that were just created.
*/
infoArea.setDefaultText(STR_FONTDLGT);
IFont tempFont(&hello);
IFontDialog::Settings fontSettings(&tempFont);
fontSettings.setTitle(IResourceId(STR_FONTDLGT));
IFontDialog fontDialog(desktopWindow(), this,
 IFontDialog::resetButton, fontSettings);
/*-----*/
If the user pressed OK in the font dialog, then change the font for
the hello IStaticText object using the tempFont object that was set
by the font dialog.
Reset the information area inactive text.
*/
if (fontDialog.pressedOK())
{
 hello.setFont(tempFont);
}
infoArea.setDefaultText(STR_INFO);
return (*this); // Return a reference to the frame
}; /* end AHelloWindow :: setHelloFont() */

```

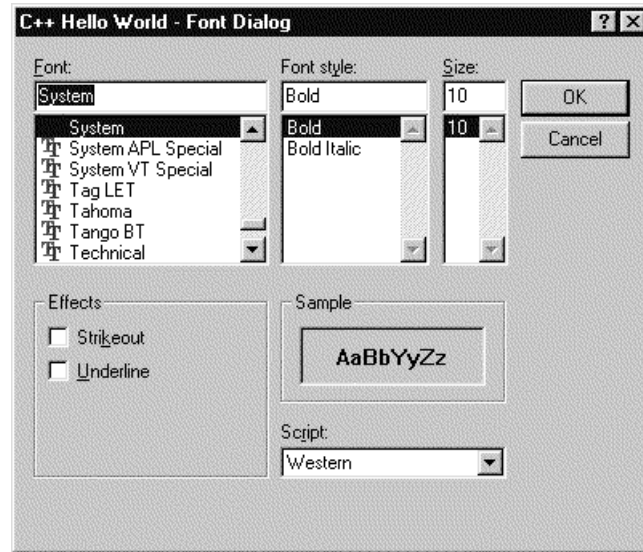
In the preceding sample, the font in an IStaticText control is changed to the font the user selects from an IFontDialog. We accomplished this by doing the following:

1. Creating an IFont object called tempFont that represents the font currently being used by the IStaticText control pointed to by hello
2. Passing a pointer to the tempFont object on the constructor to an IFontDialog::Settings object called fsettings
3. Passing the fsettings object on the IFontDialog constructor

Because fontSettings is constructed using tempFont, the IFontDialog initially displays the name, style, size, and emphasis associated with tempFont (for example, the font currently used by the IStaticText object). If the user dismisses the IFontDialog by pressing **OK**, then tempFont automatically updates itself to reflect the font the user chose via the IFontDialog. The setFont member function can be used to actually change the font of the IStaticText control to tempFont.

The following shows an example of a font dialog:





#### RELATED CONCEPTS

- “Chapter 18. Fonts” on page 271
- “Chapter 17. Common Dialogs” on page 261
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443
- “Specify Font Dialog Information” on page 266
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445



---

## Chapter 18. Fonts

A *font* is a collection of characters (both textual and symbolic) that share common visual elements between all of the characters of one size of one type style. These visual elements can be broken up into several categories which uniquely identify the font. The major categories are the name of the typeface, the size of type, the style of type, and the weight of type.

A *typeface* is a unique name, usually registered, which identifies a font by the attributes of the characters in the font. Different parts of a type character such as stroke width, presence or lack of serifs, vertex shaping, are all used to distinguish one font from another. The size of type is measured in points. A point is 1/72 of an inch, so there are 72 points in one inch. The style of type typically indicates whether a font is upright or slanted, while the weight of type refers to how light or heavy the type characters are.

Most operating systems provide a default set of fonts for you to use in your applications. These fonts are shipped with the system but they vary from one operating system type to another. It is important to keep this in mind when writing portable applications because the font you select to use on one operating system may not be available on another.

### RELATED CONCEPTS

“Chapter 17. Common Dialogs” on page 261

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

### RELATED TASKS

“Construct Fonts”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Construct Fonts

The Open Class Library provides the `IFont` class to allow you to create and manipulate fonts within your application. This section demonstrates how you can construct and modify fonts using `IFont`.

There are four ways you can construct `IFont` objects in your application:

- Create default `IFont` objects
- Create `IFont` objects using an existing window's font
- Create `IFont` objects with a specific font facename and optional attributes
- Create `IFont` objects using an existing graphic context

### Creating a Default `IFont` Object

The first way to create an `IFont` object is using the default constructor. This causes an object to be created that represents the system default font for the operating system.

> WIN

This represents the stock font object SYSTEM\_FONT.

> OS/2

This represents the System Proportional font.

> AIX

This represents the "fixed" font.

The following example shows how to construct a default IFont object:

```
IFont defaultFont();
```

### Creating an IFont Object Using a Window's Font

The second way to create an IFont object is to use a font that is currently in use by an existing window. The window containing the font is queried for the font that is in use, and the characteristics of that font are used to create another font with the same name, size and attributes. The following example demonstrates this by first creating an information area object for a frame window and then creating an IFont object from that info area. Finally, the maximum character height of the font is retrieved for this font and is used to set the height of the information area.

```
IInfoArea* infoArea = new IInfoArea(this, WND_INFO);
IFont infoFont(infoArea);
setExtensionSize(&infoArea, infoFont.maxCharHeight());
```

### Creating an IFont Object with a Specific Name

The third way to create an IFont object is to specify an exact facename of a font along with some optional characteristics of the font such as size, type information, and graphic context. Because individual font facenames may not be available on all operating systems, this approach may not be portable across operating systems.

The highlighted lines in the following example show you how to create a font with a specific name and point size, and then how to change the point size of the text associated with visible Open Class Library objects. A Helvetica 8 point font is initially constructed. It is then modified to be a Helvetica 12 point font and applied to the title. It is then modified again to be a Helvetica 20 point font and is applied to the checkbox.

```
#include <ifont.hpp>
IFont font("Helvetica",8);
title1.setAlignment(IStaticText::centerLeft);
title1.setText(STR_TITLE1);
font.setPointSize(12);
title1.setFont(font);
check1.setText(STR_CHECK1);
font.setPointSize(20);
check1.setFont(font);
```

To test the font statements, include the highlighted lines in the amcelcv.cpp file. The amcelcv.cpp file, from the Multicell Canvas Sample, is located in the samples/ioc/mcelcv directory.

Note that the example above demonstrates dynamically creating new fonts from existing fonts using different IFont member functions to change the characteristics of the font. There are a wide range of member functions available to use to change attributes of the font. The following is a list of the more common font attributes that can be modified:

| Attribute            | Description                                                                                                                                                                                                                                                                                                                |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Font facename        | The setName member function is used to modify the exact facename of the IFont                                                                                                                                                                                                                                              |
| Font pointsize       | The setPointSize member function is used to modify the point size of the font.                                                                                                                                                                                                                                             |
| Font characteristics | The setUnderscore, setStrikeout, setBold, setItalic and setOutline member functions are used to modify a specific attribute of the font. These functions underline the text, draw a line through the text, change the font to have more weight, slant the font, and only draw the outline of the characters, respectfully. |

For portable applications, consider using a font dialog.

### Creating an IFont Object from a Graphic Context

The final way to construct an IFont object is to use an existing graphic context, represented by an IPresSpaceHandle. The graphic context is queried to determine the font that is currently in use and that font is then used to construct an IFont object with the same name and characteristics. The following example demonstrates this:

```
IPresSpaceHandle ps = IPresSpaceHandle();
IFont tempFont(ps);
```

#### RELATED CONCEPTS

- “Chapter 18. Fonts” on page 271
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Chapter 17. Common Dialogs” on page 261
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445
- Multicell Canvas Sample

#### RELATED REFERENCES

- IFontDialog
- IMGraphic
- IText



---

## Chapter 19. Help Information

*Help information* is the information about how to use an application. By describing an application's choices, objects, and interaction techniques, help information can assist users in learning to use a product.

The `IHelpWindow` only addresses help panels. You do not need this class to display other forms of help such as fly-over help and information areas.

The `IHelpWindow` class lets you use either native Windows help (Rich Text Format) or the OS/2 Information Presentation Facility (IPF) to provide help information for applications:

- 

**AIX**

**OS/2**

You use IPF help files and the default style is set to `IHelpWindow::ipfCompatible`. This default cannot be changed.

- 

**WIN**

You have the option to use Native Windows help or IPF help. The default is to use Native Windows help. To use IPF help, you call `IHelpWindow::setDefaultStyle(IHelpWindow::ipfCompatible)` before constructing `IHelpWindow` objects. You can also specify the style `IHelpWindow::ipfCompatible` when constructing an `IHelpWindow` object.

The RTF and IPF help formats determine how you code (tag) the source files for your help panels, the tool you use to build them into binary help files, the tool you can use to view them, and the look of the help panels when they are displayed to the user.

To view an IPF help file on Windows, use **`iview.exe`**. On OS/2, the executable is **`view.exe`** and on Motif it is **`xview`**.

You create and associate an `IHelpWindow` object with one of your application's main windows. The Open Class Class Library also provides an `IHelpHandler` class to process help window events. When you associate an application window with a help window, help events are dispatched to the help handlers attached to the application window.

To use the `IHelpWindow` class, you create and associate an `IHelpWindow` object with one or more of your application's main windows. The help panels displayed by `IHelpWindow` are written and compiled using tools native to your development environment or provided with the VisualAge C++ product. To present a consistent look and feel for help across development environments, use the IPF help information compiler and runtime.

**WIN**

On the Windows environment, you can also display your information using the native Windows help facility. Although the process of generating the help

information is different, and the display and interaction of the Windows help facility is different from IPF, you can still use the classes of IHelpWindow to interact with native Windows help. However, some of the IHelpWindow functions have no equivalent for Windows help and therefore return without performing any operation. To use IPF help files, call the following before constructing your IHelpWindow objects:

```
IHelpWindow::setDefaultStyle(IHelpWindow::ipfCompatible);
```

You can also specify the style when calling the IHelpWindow constructor.

### Overriding the Behavior of the Help Facility

In some cases, you may want to override the behavior of the help facility. You can attach an IHelpHandler object to a frame window that has been associated with a help window. By default, the IHelpHandler provides meaningful message boxes when odd situations occur, such as the help file not being found. To override this behavior, derive your own class from IHelpHandler and override the functions you need to change.

See the Hello World Sample Version 5 Sample for a demonstration of the IHelpHandler class. You can find this sample in the samples/ioc/hello5 directory.

#### RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Create Help Information”

“Add Fly-Over Help” on page 282

“Create an Information Area” on page 9

“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Create Help Information

Use the following steps to create IPF help information for your application:

- Create a file containing the help information.

Create the source text that displays in your application’s help window using the IPF format (.ipf file) for the OS/2 operating system and the IPF/X format (.ipf file) for the Windows and Motif environments. Compile your file into a help file (.hlp file) using the IPFC compiler.

Refer to the Information Presentation Facility documentation for descriptions of the tags you use to create the source .ipf file.

For an example of a .ipf source file, see the Hello World version 5 ahellow5.ipf file, which is described in Hello World Version 5: Adding Canvases, a List Box, Native System Functions, and.

- Define the help window title and the help submenu in your resource file. The following is from the Windows ahellow5.rc file:

```
STR_HTITLE, "C++ Hello World - Help Window" //Help window title string
POPUP "&Help" , MI_HELP , RIGHTJUSTIFY
BEGIN
```



```

 MENUITEM "&General help..." , SC_HELPEXTENDED /*MIS_SYSCOMMAND*/
 MENUITEM "&Keys help..." , SC_HELPKEYS /*MIS_SYSCOMMAND*/
 MENUITEM "Help &index..." , SC_HELPINDEX /*MIS_SYSCOMMAND*/
END

```

The AIX and OS/2 .rc file appears as follows:

```

STR_HTITLE, "C++ Hello World - Help Window"
SUBMENU " Help", MI_HELP, MIS_HELP
BEGIN
 MENUITEM "_General help...", SC_HELPEXTENDED, MIS_SYSCOMMAND
 MENUITEM " _Keys help...", SC_HELPKEYS, MIS_SYSCOMMAND
 MENUITEM "Help _index...", SC_HELPINDEX, MIS_SYSCOMMAND
END

```

Applications with .rc files that use the following constants should include the icconst.h file:

- SC\_HELPKEYS
- SC\_HELPINDEX
- SC\_HELPEXTENDED

MI\_HELP is the help menu ID.

Normally, you specify MIS\_HELP for a menu item to cause a help event, rather than a command event, to be posted when the menu item is selected.

When you specify MIS\_HELP (OS/2) or RIGHTJUSTIFY (Windows) for a submenu item, Motif automatically positions the item to the far right of the menu bar. OS/2 PM ignores MIS\_HELP specified on submenu items.

When MIS\_SYSCOMMAND is specified with the predefined SC\_HELP\* IDs, a system command event is generated. The default system command handler recognizes the predefined IDs and shows the appropriate help panel, except for SC\_HELPKEYS, which by default does nothing. SC\_HELP\* IDs are defined in the <icconst.h> file. You can override this default processing for SC\_HELPKEYS, using an IHelpHandler, which is described in a later step.

- Define a help table in the resource file.

The help table defines the relationship between the window ID and the general or contextual panel ID that is defined in the .ipf file. The following help table is defined in the resource file, ahellow5.rc, for Hello World version 5:

```

HELPTABLE HELP_TABLE
BEGIN
 HELPIITEM WND_MAIN, SUBTABLE_MAIN, 100
 HELPIITEM WND_TEXTDIALOG, SUBTABLE_DIALOG, 200
END
HELPSUBTABLE SUBTABLE_MAIN //Main window help subtable
BEGIN
 HELPSUBITEM WND_HELLO, 100 //Hello static text help ID
 HELPSUBITEM WND_LISTBOX,102 //List box help ID
 HELPSUBITEM MI_EDIT, 110 //Edit menu item help ID
 HELPSUBITEM MI_ALIGNMENT, 111 //Alignment menu item help ID
 HELPSUBITEM MI_LEFT, 112 //Left command help ID
 HELPSUBITEM MI_CENTER, 113 //Center command help ID
 HELPSUBITEM MI_RIGHT, 114 //Right command help ID
 HELPSUBITEM MI_TEXT, 199 //Text command help ID
END //
HELPSUBTABLE SUBTABLE_DIALOG //Text dialog help subtable
BEGIN //
 HELPSUBITEM DID_ENTRY, 201 //Entry field help ID

```

```

HELPSUBITEM DID_OK, 202 //OK command help ID
HELPSUBITEM DID_CANCEL, 203 //Cancel command help ID
END //

```

WND\_HELLO and WND\_LISTBOX are control IDs, MI\_\* are menu item IDs, and the DID\_\* are push button IDs. Each window ID is related to a help panel ID. In the preceding example, WND\_MAIN and WND\_HELLO both correspond to help panel ID 100. That is, pressing the **F1** key in the main window area displays the same help panel as selecting **General help...** from the **Help** submenu.

- Create a help window object for your application window. Use the IHelpWindow class to associate help information with an application window. Hello World version 5 defines the private data member, helpWindow, as an IHelpWindow object. It is initialized in the AHelloWindow constructor in ahellow5.cpp using the following statement:

```
helpWindow(HELP_TABLE, this)
```

The first parameter identifies the help table defined in the resource file. The second parameter identifies the application window for which the help is being provided.

Use the IHelpWindow::addLibraries member function to associate a help file with a help window. The AHelloWindow constructor in Hello World version 5 provides an example:

```
helpWindow.addLibraries("ahellow5.hlp");
```

You can set the title of the help window by using the IHelpWindow::setTitle member function. The following code sets the title from a string defined in the resource file:

```
helpWindow.setTitle(STR_HTITLE);
```

- Create a help handler by creating an IHelpHandler derived class. A help handler processes help events. Create your own help handler class derived from IHelpHandler to provide help event processing that is unique to your application. Hello World version 5 uses a help handler only to display the keys help panel describing accelerator key definitions.

To override keys help processing, do the following:

1. Define a new class that is derived from IHelpHandler, as shown in the Hello World version 5 class definition file, ahellow5.hpp.

```

class AHelpHandler : public IHelpHandler
{
public:
/*----- Destructor -----|
| Destruct the object with: |
| 1) No parameters |
-----*/
 virtual ~AHelpHandler() { }
protected:
/*----- Override keysHelpId Function -----|
| The keysHelpId() function is called to set the event result to the |
| ID within the Hello World help libraries for Keys Help. |
-----*/
 virtual bool
 keysHelpId(IEvent& evt);
};

```

2. Provide the overridden virtual function `keysHelpId`, which is called when keys help is requested. The following code, from the Hello World version 5 `ahellow5.cpp` file, shows how to implement this function.

```
bool AHelpHandler :: keysHelpId(IEvent& evt)
{
 evt.setResult(1000); //1000=keys help ID in
 // ahellow5.ipf file
 return (true); //Event is always processed
} /* end AHelpHandler :: keysHelpId(...) */
```

In the preceding code, the help panel ID for the Hello World version 5 keys help is set in the event result.

3. Start and stop help events processing.

Your help handler, previously described, does not begin handling help events until you use the `handleEventsFor` member function. For example, the following code causes the `helpHandler` to begin processing help events for this frame window:

```
helpHandler.handleEventsFor(this);
```

Typically, you include this statement in the constructor for the frame window.

Note that the window that handles help events must be an associated window. That is, you should identify the window as the associated window on the `IHelpWindow` constructor or explicitly identify the window as an associated window using the `IHelpWindow::setAssociatedWindow` function.

When you want to stop handling help events, for example, when you close your frame window, use the `stopHandlingEventsFor` member function, as follows:

```
helpHandler.stopHandlingEventsFor(this);
```

You typically include this statement in the destructor for the frame window.

4. Associate secondary frame windows with the parent window's help window.
5. Attach the following special handler to child frame windows in your application. This handler is needed so that help processes correctly for these windows.

```
class ChildFrameHelpHandler : public IHandler {
typedef IHandler Inherited;
/*****
 * This handler enables the OS/2 Help Manager to use help tables to display
 * contextual help for a child frame window (one whose parent window is not
 * the desktop). This handler should only be attached to child frame windows.
 *****/
public:
virtual ChildFrameHelpHandler
 &handleEventsFor (IFrameWindow* frame),
 &stopHandlingEventsFor (IFrameWindow* frame);
protected:
virtual bool
 dispatchHandlerEvent (IEvent& evt);
ChildFrameHelpHandler
 &setActiveWindow (IEvent& evt, bool active = true);
private:
virtual IHandler
 &handleEventsFor (IWindow* window),
 &stopHandlingEventsFor (IWindow* window);
};
```

6. In the .cpp file, do the following:

```
bool ChildFrameHelpHandler :: dispatchHandlerEvent (IEvent& evt)
{
 switch (evt.eventId())
 {
 case WM_ACTIVATE:
 setActiveWindow(evt, evt.parameter1().number1());
 break;
 case WM_HELP:
 setActiveWindow(evt, true);
 break;
 default:
 break;
 } /* endswitch */
 return false; // Never stop processing of event
}
ChildFrameHelpHandler&
ChildFrameHelpHandler :: setActiveWindow (IEvent& evt,
 bool active)
{
 IHelpWindow* help = IHelpWindow::helpWindow(evt.window());
 if (help)
 {
 IFrameWindow* frame = 0;
 if (active)
 {
 frame = (IFrameWindow*)evt.window();
 }
 help->setActiveWindow(frame, frame);
 }
 return *this;
}
ChildFrameHelpHandler&
ChildFrameHelpHandler :: handleEventsFor (IFrameWindow* frame)
{
 IASSERTPARAM(frame != 0);
 Inherited::handleEventsFor(frame);
 return *this;
}
ChildFrameHelpHandler&
ChildFrameHelpHandler :: stopHandlingEventsFor (IFrameWindow* frame)
{
 IASSERTPARAM(frame != 0);
 Inherited::stopHandlingEventsFor(frame);
 return *this;
}
IHandler& ChildFrameHelpHandler :: handleEventsFor (IWindow* window)
{
 // private to hide version in IHandler
 ITHROWLIBRARYERROR(IC_MEMBER_ACCESS_ERROR,
 IErrorInfo::invalidRequest,
 IException::recoverable);

 return *this;
}
IHandler& ChildFrameHelpHandler :: stopHandlingEventsFor (IWindow* window)
{
 // private to hide version in IHandler
 ITHROWLIBRARYERROR(IC_MEMBER_ACCESS_ERROR,
 IErrorInfo::invalidRequest,
 IException::recoverable);

 return *this;
}
}
```

### Assigning Help IDs to Windows

You can assign helpIDs directly to windows by using `IWindow::setHelpId( unsigned long topicId )`; while maintaining portability. When F1 is pressed, the window with focus is then be queried for its context helpID and if one is found, it

is used to identify the help panel to display. If one is not found and there is a help table, it is searched for the appropriate helpId to use.

If you are not using a help table, the helpId of the active frame is used as the general help panel.

### Using Native Windows Help Files with IHelpWindow

#### WIN

By default, IHelpWindow uses the native help for the platform to handle help requests using a .hlp file that was created from RTF or IPF source. This allows the code you write to be portable while giving your users native look-and-feel. IPF help files are also portable between the various platforms. RTF help files are not portable. In addition, some functions provided by IPF are not available when using Windows native help (RTF). The major differences are as follows:

- There is only one instance of Windows Help for all your applications.
- The size and position of WinHelp is NOT relative to any particular application.
- IPF uses a set of filenames to locate help; WinHelp is one file at a time, but you must still use IHelpWindow::addLibraries to change the help file name.
- IPF help has sections for CONTENTS, INDEX, KEYS, GENERAL, and USING. WinHelp has HELP\_FINDER and HELPONHELP.
- Only IPF allows the application to change the Help Manager menu and accelerators.
- Only IPF allows the application to set a title for each help object.
- Only IPF allows querying of handles for IPF sections, such as the coverage or contents.
- Only IPF allows the application to toggle showing of the help IDs in the help manager window titles.

#### RELATED CONCEPTS

“Chapter 19. Help Information” on page 275  
“Chapter 1. Windows” on page 1  
“Chapter 1. Windows” on page 1  
“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Add Fly-Over Help” on page 282  
“Add Menus to Your Application” on page 57  
“Add a Menu Bar” on page 59  
“Write an Event Handler” on page 25  
Hello World Version 5: Adding Canvases, a List Box, Native System Functions, and Help  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

IFlyText  
IFlyOverHelpHandler  
IHelpHandler  
IHelpWindow

## Add Fly-Over Help

*Fly-over help* displays short help windows for the object that the mouse pointer is currently positioned over. As users move their mouse pointer over various objects, different help windows are displayed. In addition, you can display descriptive text for the object in a text control, such as the information area at the bottom of the window.

You can use the following classes to add fly-over help to your applications:

### IFlyText

Use this static text control to display brief informative messages for a window, such as the function of a push button contained in a toolbar.

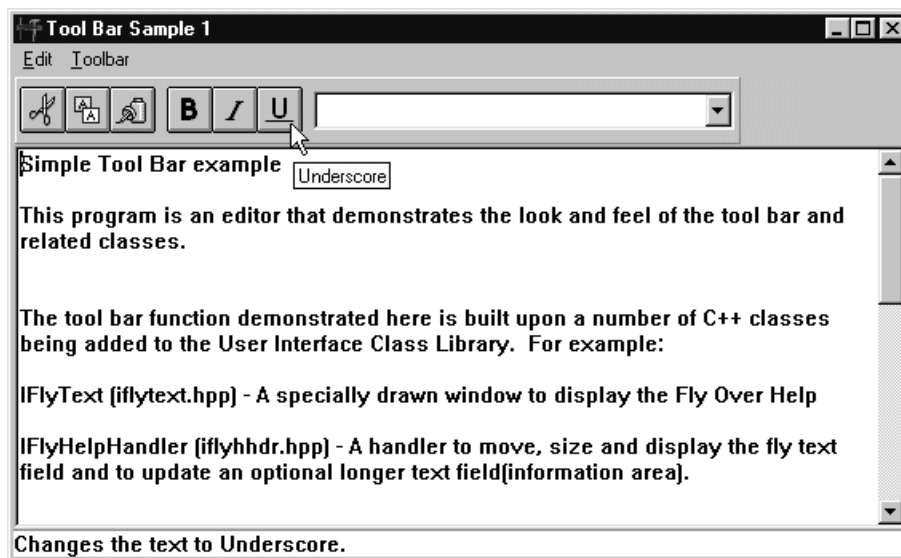
### ITextControl

Use a class derived from this base class to display longer, more detailed text. Typically, you use an information area as the ITextControl.

### IFlyOverHelpHandler

Attach an IFlyOverHelpHandler to a window to provide context-specific help messages for any window that is a child of the window you attached the handler to.

The following figure shows an example of fly-over help used on a toolbar.



### Displaying Fly-Over Help Information

The IFlyText control displays help messages in a bordered window that is sized large enough to contain the help text. The text is displayed using one row that does not re-flow. The default font used to display the help messages in PM is 8-point Helvetica.

The IFlyText control positions itself relative to one of the corners of the window the mouse pointer is over. Fly-over help displays the help message so that none of the text is drawn outside the desktop. This determines the corner of the IFlyText control from which it is drawn. The order in which the IFlyText control attempts to position itself is as follows:

1. Lower-right corner

2. Lower-left corner
3. Upper-left corner
4. Upper-right corner

You can construct instances of this class from a given window ID and an owner window.

For example:

```
IFrameWindow frameWindow(0x1000);
IToolBar toolbar(0x1001, &frameWindow, &frameWindow);
IFlyText flyText(0x1002, &toolbar);
IFlyOverHelpHandler flyOverHelpHandler(&flyText);
flyOverHelpHandler.handleEventsFor(&toolbar);
```

IFlyText provides the function `setText` to set the accessible attributes of each instance, and it provides functions `setRelativeWindowRect` and `relativeWindowRect` to set and query the position relative to the IFlyText control.

#### WIN

For Windows, the actual owner of the control is set to the first IFrameWindow object found in the owner chain of the specified owner. This insures proper operation in cases where the parentage of the specified owner is changed. For example, this occurs when you make a toolbar a floating toolbar.

The control ID of IFlyText is ignored.

#### Attach Handlers to Provide Context-Sensitive Help

By attaching an IFlyOverHelpHandler to a window, you can provide context-specific help messages for any window that is a child of the window you attached the handler to.

You can use the IFlyOverHelpHandler to update an IFlyText control, an ITextControl, or both. The IFlyText control contains short messages (one or two words, for example) for a window, and ITextControl displays more descriptive text in the information area. You do not need a string associated with every window in your application. When a help string cannot be found for a window, a single blank is displayed by default to keep the frame extension handler from hiding the ITextControl when it contains a null string. You can also use the `setMissingText` function to set the text to be displayed when an information string cannot be found.

Note that new-line characters are removed from a string before they are displayed in the IFlyText control.

The last two parameters of each handler constructor are time delays expressed in milliseconds. The first delay, indicates the time the mouse pointer must remain in the same location before the fly-over help is displayed for the first time. The second delay, indicates the time the mouse pointer must remain in the same location after the fly-over help has been displayed for the first time.

You can change the length of the first delay using the `setInitialDelayTime` member function or query what is currently set using the `initialDelayTime` member function. Use the `setDelayTime` member function to change the second delay and use `delayTime` to query what is currently set for that second delay.

You associate context-specific help for a window to a help message by specifying a window identifier. This identifier is used with an offset to load strings from a string table. Specify different offsets into the string table for the IFlyText and the ITextControl objects to display different help messages in each of the controls.

Note that to display help for a window using the string, you must either create the window using the Open Class Library or wrapper an existing window.

### **Dynamically Add Help Text to Windows**

You can also dynamically associate help text to a window using IFlyOverHelpHandler, a window handler. This is useful when you dynamically add controls to a canvas or push buttons to a tool bar.

Use the following functions to dynamically add or remove the help text specified for a window:

#### **flyHelpText**

Returns the short help text for a window if you have dynamically added help text for the window.

#### **longHelpText**

Returns the long help text for a window if you have dynamically added long help text for the window.

#### **setHelpText**

Sets help text for a window by specifying a string or resource ID. 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.

#### **removeHelpText**

Removes help text you added to a window through the setHelpText function.

#### **RELATED CONCEPTS**

“Chapter 19. Help Information” on page 275

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### **RELATED TASKS**

“Create Help Information” on page 276

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445



---

## Chapter 20. Clipboard

A *clipboard* is a system-wide place for users to store data temporarily. The clipboard enables your user to move data within a single application or to exchange data among applications. Typically, a user selects data in the application using the mouse or keyboard, then initiates a cut or copy operation on the selected data. The clipboard can hold an entire object or part of that object, and it can hold any kind of object. For example, the clipboard can hold a single line of text or an entire database, a single line segment or an entire graphic.

When the user selects the paste operation, the data is transferred to the application from the clipboard.

Only a single item of data can be stored in the clipboard at a time. Therefore, do not use the clipboard to store data unless a user requests it because you can overlay the user's data stored there. This is important: the user must always control access to the clipboard.

While you can only store a single item of data in the clipboard, you can store this item in multiple formats. This allows an application to choose the format it supports that gives it the most information about the data. For example, a graphics application might copy a picture into the clipboard as both a metafile and a bitmap. This allows applications that support both metafiles and bitmaps to retrieve the picture as a metafile if it needs to modify the picture or as a bitmap if it only needs to display the picture.

IClipboard predefines several system clipboard formats. In addition, any application can create and register additional private formats.

Before you can write any data to, or read any data from, the clipboard, you must first open it. Only a single application at a time can open the clipboard. If an application tries to open the clipboard but another application already has it open, it waits until the clipboard is available. The default behavior of the clipboard classes minimizes the time the clipboard is open.

If you use the default behavior of IClipboard, the clipboard functions that require an open clipboard open it when needed and close it when finished. You turn off the default behavior of IClipboard when you explicitly open the clipboard by calling IClipboard::open. If you open the clipboard in this manner, functions in IClipboard will not close the clipboard when complete. If you explicitly open the clipboard, you must close the clipboard by calling IClipboard::close. You can turn off the default behavior of IClipboard to place different formats of your data on the clipboard without opening and closing it to write each format.

All clipboard operations must be associated with a window. You provide this window on the IClipboard constructor. If necessary, IClipboard makes this window the *owner* of the clipboard. The clipboard owner is the window responsible for the data put on the clipboard. It is also the window that the operating system sends messages to for events relating to the clipboard. The IClipboard object establishes this window as the system clipboard owner when you call IClipboard::empty. If you call IClipboard::owner before calling empty, your window will not be returned because it is not yet the system clipboard owner.

The clipboard classes support an advanced concept called *delayed rendering*. Delayed rendering allows you to wait until another application requests the data before you put the data on the clipboard. You activate delayed rendering by supplying 0 for the data when you call the clipboard functions to place data on the clipboard.

You process clipboard events by creating and attaching an IClipboardHandler object to your clipboard owner window. In particular, if you use delayed rendering, you must attach an IClipboardHandler object to your clipboard's window (the owner window). The window dispatcher calls this handler when a request is made to the clipboard for data that has not been placed there yet.

Because the clipboard should only be kept open for a short time, create IClipboard objects as temporary objects with a short lifetime. This helps ensure that the clipboard is only open for the time necessary.

The IClipboard destructor always closes the clipboard if it is still open.

#### RELATED CONCEPTS

"Chapter 23. Direct Manipulation" on page 313  
"Chapter 1. Windows" on page 1  
"Chapter 3. Events and Event Handlers" on page 19

#### RELATED TASKS

"Add Clipboard Support"  
"Sample: Add Clipboard Support" on page 289  
"Use Default Direct Manipulation" on page 316  
"Add Images to Drag Items" on page 331  
"Write an Event Handler" on page 25  
"Task and Samples Cross-Reference Table" on page 445

---

## Add Clipboard Support

You provide the clipboard owner on the IClipboard constructor and allow the functions needing an open clipboard to open the clipboard and close it when finished.

The following example uses an IClipboard object to copy text from an MLE into the clipboard and then to paste the data from the clipboard back into the MLE:

```
bool CommandHandler::command (ICommandEvent& event)
{
 switch(event.commandId())
 {
 case MI_COPY:
 {
 IClipboard clipboard(event.window()->handle());
 clipboard.empty();
 clipboard.setText(edit.selectedText());
 return true;
 }
 case MI_PASTE:
 {
 IClipboard clipboard(event.window()->handle());
 if (clipboard.hasText())
 edit.add(clipboard.text());
 return true;
 }
 }
}
```

```

 }
 }
 return false;
}

```

An application can put only one item of one format into the clipboard. You can only put multiple items into the clipboard if each has a different format. Adding multiple items with the same format results in replacing the data.

Use the classes described below to create and manage a clipboard for your application:

| Class              | Description                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IClipboard         | Interface declaration class that creates a clipboard object.                                                                                                                                 |
| IClipboard::Cursor | Nested class that iterates the available formats of data in the clipboard.                                                                                                                   |
| IClipboardHandler  | Handler class to process the events that the clipboard sends to its owner. This includes requests to render clipboard data for formats that are put on the clipboard with delayed rendering. |

To clear the contents of the clipboard, use the `IClipboard::empty` member function. This empties the contents of the clipboard and establishes the owner provided on `open` as the real clipboard owner. This function opens the clipboard if it is not already open and closes it after use unless you have explicitly opened the clipboard by calling `open`.

Use the `IClipboard::isOpen` function to query the clipboard status. It returns true if the clipboard is open.

### Move Data Using the Clipboard

You can use the following `IClipboard` class member functions to move data to and from the clipboard:

| Function                 | Description                                                                                                                                                                                                                                                          |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>setText()</code>   | Copies the passed text into shared memory and places it on the clipboard with the format <code>IClipboard::textFormat()</code> .                                                                                                                                     |
| <code>setBitmap()</code> | Copies the passed bitmap and places the handle on the clipboard with the format <code>IClipboard::bitmapFormat()</code> .                                                                                                                                            |
| <code>setData()</code>   | Copies the passed data buffer and places it on the clipboard with the format specified. Register any private formats first by calling <code>registerFormat</code> . If "data" is 0, create an <code>IClipboardHandler</code> to process requests to render the data. |
| <code>setHandle()</code> | Places the passed handle on the clipboard.                                                                                                                                                                                                                           |
| <code>hasText()</code>   | Returns true if the clipboard has data with the format <code>IClipboard::textFormat()</code> .                                                                                                                                                                       |
| <code>hasBitmap()</code> | Returns true if the clipboard has data with the format <code>IClipboard::bitmapFormat()</code> .                                                                                                                                                                     |

| Function  | Description                                                                                                                                                                                                                                                                                                                             |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hasData() | Returns true if the clipboard has data of any format.                                                                                                                                                                                                                                                                                   |
| text()    | Returns data of the format <code>IClipboard::textFormat()</code> as an <code>IString</code> object.                                                                                                                                                                                                                                     |
| bitmap()  | Copies data of the format <code>IClipboard::bitmapFormat()</code> and returns an <code>IBitmapHandle</code> .                                                                                                                                                                                                                           |
| data()    | Returns a <code>void*</code> value. This value can either be a pointer to the data being rendered or a handle depending on the format of the data. This function always leaves the clipboard open. The caller must copy the data, if necessary, before closing the clipboard. Access to the data is lost after the clipboard is closed. |

You can request delayed rendering of data by using the `setData()` member function with the appropriate format and a 0 data pointer. Delayed rendering requires that you create an `IClipboard` handler to process requests to render the data when needed.

**> AIX**

While all three environments support delayed rendering by allowing an application to pass a zero pointer for the data, Motif differs in that it passes back a data identifier that needs to be used when the data is later placed on the clipboard. Further, actually putting the data on the clipboard requires a different api (`XmClipboardCopyByName`) then putting the data on the clipboard normally (`XmClipboardCopy`). OS/2 and Windows use the same APIs in both scenarios. This API is handled in Open Class Library with a call to `IClipboard::setData()`.

**> AIX**

Open Class now has the member `IClipboardHandler::setDelayedData()`. On Motif, implement this member by calling `IClipboardImp::setDelayedData()`. The Motif version of this implementation function, `IMotifClipboardImp::setDelayedData()`, calls `XmClipboardCopyByName`. The OS/2 and Windows versions of `setDelayedData()` calls `setData()`.

**> AIX**

Code that needs to be portable to Motif must change the override of `IClipboardHandler::renderFormat()` from a call to `IClipboard::setData()` to `IClipboardHandler::setDelayedData()`.

When you copy data to the clipboard in response to a delayed rendering message on OS/2 and Windows, the data is actually placed on the clipboard. Repeated attempts to access the data, return the data on the clipboard. No further action is necessary from the owner of the data. On Motif, every time delayed-rendered data is requested, it is acquired from the owner's callback routine. It acts like the data never ends up on the clipboard. This is a problem in scenarios where an object is cut from an application because the object is deleted at the time of the cut and is

therefore unavailable at the time the application is asked to provide the data in a delayed-rendering scenario. This is a Motif restriction and your code will have to be different for Cut than it is in Copy.

`IClipboardHandler` processes the clipboard event by creating an `IEventobject` and routing it to the appropriate virtual function. The virtual function allows you to supply your own specialized processing of the event. The return values from the virtual function specify whether the paint event is passed on to another handler object to be processed.

The `dispatchHandlerEvent` member function evaluates the event to determine if it is appropriate for this handler object to process. If it is, this function calls the virtual function used to process the event.

#### RELATED CONCEPTS

“Chapter 20. Clipboard” on page 285

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Use Default Direct Manipulation” on page 316

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

`IClipboardHandler`

---

## Sample: Add Clipboard Support

This sample demonstrates how you can add clipboard support to a control, such as a container, that does not have built-in clipboard support. It also demonstrates how to use delayed rendering to put the data on the clipboard only when a user requests it during a paste operation. You can find the following, from the Clipboard Sample, in the directory `samples/ioc/clipbrd`.

In the `clipbrd.hpp` file we do the following:

1. Declare a `Department` container object with an interface for rendering the object into a predefined format string and for initializing an existing object from a format string. This enables us to save the state of a `Department` object on a copy or cut operation so that we can create new `Department` objects during a paste operation.
2. Declare a `ContainerCutPasteHandler` object that is responsible for the following:
  - Creating and displaying a popup menu containing cut, copy, and paste choices.
  - Responding to user commands to cut, copy, and paste the content of `Department` objects on the clipboard.
  - Responding to clipboard handler requests to render the data to the clipboard.

```
#ifndef _CLIPBRD_
#define _CLIPBRD_
#include <icliphdr.hpp>
#include <icmdhdr.hpp>
#include <istring.hpp>
#include <icnr.hpp>
```

```

//*****
// Class: Department *
// *
// Purpose: Defines the data stored in the container for a Department. *
// *
//*****
class Department : public IContainerObject {
public:
 Department (const IString& name=IString(),
 const IString& address=IString())
 : IContainerObject (name),
 strAddress(address) {}
// Add functions to query and set the data.
virtual IString
 name () const,
 address () const;
virtual Department
 &setName (const IString& name),
 &setAddress (const IString& address);
// Define the functions to render an object both as a
// private format and as a normal text string, and to
// re-construct the object from the private format.
IString
 asString () const,
 text () const;
Department
 &initializeFromString (const IString& renderedString);
// Define the separator character (a tilde) that separates
// the fields of the object in its string format.
static const IString
 separator,
 renderedFormat;
// Define a function to return the offset of the Address field.
static unsigned long
 offsetOfAddress () { return ICONAINTERCOLUMN_OFFSETOF(Department, strAddress); }
private:
IString
 strAddress;
};
class ICnrObjectSet;
//*****
// Class: ContainerCutPasteHandler *
// *
// Purpose: Adds Clipboard support to the container for a Department *
// object. This includes: *
// 1) A container menu handler to show a popup menu with *
// cut, copy, and paste choices. *
// 2) A command handler to process the cut, copy, and paste *
// requests. *
// 3) A clipboard handler to process requests from the clipboard *
// to render data not yet on the clipboard. *
//*****
class ContainerCutPasteHandler : public ICommandHandler,
 public ICnrMenuHandler,
 public IClipboardHandler {
public:
 ContainerCutPasteHandler (IContainerControl& container);
 ContainerCutPasteHandler ();
IContainerControl
 &container () { return cnr; }
protected:
// Define the command handler callback.
virtual bool
 command (ICommandEvent& event);
// Define the popup menu callback.
bool
 makePopUpMenu(IMenuEvent& cnEvt);

```

```

// Define the callbacks to render data on the
// clipboard.
virtual bool
 clipboardEmptied (IEvent& event),
 renderFormat (IEvent& event,
 const IString& format),
 renderAllFormats (IEvent& event);
// Define a string object to use as a separator between fields
// for the private format.
static const IString
 separator;
IString
 selectedData ();
private:
IContainerControl
 &cnr;
ICnrObjectSet
 *objectList;
};
#endif

```

To support copying our Department object to the clipboard, we have devised a design to render a Department object as a data string and to create and initialize a new Developer object from data stored in such a data string. This data string contains fields separated by a character that we know does not exist in the data of our Department object. We define this separator character and initialize it to a tilde character later in clipbrd.cpp.

The ContainerCutPasteHandler utilizes a very similar design to store a series of container objects on the clipboard. The ContainerCutPasteHandler must use a different separator to distinguish the parts of its data. In the .cpp, we define the caret (^) character as the separator between objects on the clipboard. When our handler receives a request to render the data to the clipboard in our private format, it creates a string with the following layout:

|        |               |         |               |         |
|--------|---------------|---------|---------------|---------|
| countn | separator (^) | object1 | separator (^) | objectn |
|--------|---------------|---------|---------------|---------|

The number of objects in the string is stored as a text number in the firstseparator delimited field of the string. Also in the string, we put Department *object1* through *objectn* in their own separator delimited format with the following layout:

|                 |               |                    |
|-----------------|---------------|--------------------|
| Department name | separator (^) | Department address |
|-----------------|---------------|--------------------|

For example, if the name field of a Department object is "Accounting" and the address field is "Building 4000," then the format string using the tilde character (~) as the separator is:

```
"Accounting~Building 4000"
```

If in addition to the Accounting department object above, we stored a "Sales" Department object with an address of "Building 5000," they would collectively appear on the clipboard as the string:

```
"2^Accounting~Building 4000^Sales~Building 5000"
```

Rather than copying our objects to the clipboard during the cut or copy operation, we have added support for delayed rendering. The design entails maintaining a collection of the objects cut or copied to the clipboard. When a user requests the data on the clipboard, our ContainerCutPasteHandler's renderFormat routine is

called to put the data on the clipboard. It iterates the objects in the collection and writes their data to the clipboard in the string format previously described.

A user that copies data to the clipboard and later pastes it into an application usually expects that the data will be the same as it was when it was first copied. To ensure this, any time the data of one of the objects in our collection changes or the object is removed from the container, we must force the delayed rendering mechanism to put the objects on the clipboard first. This support does not exist in our current clipboard sample.

This demonstrates that while delayed rendering has the potential for improving the performance of your application, it also increases its complexity. We therefore recommend that you first determine that you need to improve performance before deciding to add delayed rendering support to your application.

In the clipbrd.cpp file we do the following:

1. Define our static Separator objects:

```
// Define our static Separator objects:
// The first separates fields in the Department object string;
// the second separates multiple Department object strings.
const IString Department::separator(" ");
const IString ContainerCutPasteHandler::separator("^");
```

2. Define the private format of our Department object:

```
// Define the private format of our Department object.
const IString Department::renderedFormat("Department_rendered");
```

3. Construct the ContainerCutPasteHandler, enable the event handlers, and register the private clipboard format.

```
/*-----
ContainerCutPasteHandler::ContainerCutPasteHandler
-----*/
Construct the handlers, register our private clipboard format, and
attach the handlers to the container.
-----*/
ContainerCutPasteHandler :: ContainerCutPasteHandler (IContainerControl& container)
: cnr(container),
 objectList(new ICnrObjectSet())
{
 // Enable the command, menu, and clipboard handlers.
 ICommandHandler::handleEventsFor(&container);
 ICnrMenuHandler::handleEventsFor(&container);
 IClipboardHandler::handleEventsFor(&container);
 // Register the Department object's private format.
 IClipboard::registerFormat(Department::renderedFormat);
}
```

4. Process the clipboard operations cut, copy, and paste.

```
/*-----
ContainerCutPasteHandler::command
-----*/
Handle the command events associated with the clipboard (Cut, Copy,
and Paste).
-----*/
bool ContainerCutPasteHandler::command (ICommandEvent& event)
{
 switch(event.commandId())
 {
 case MI_CUT :
 case MI_COPY :
 {
 // Create a clipboard object.
 IClipboard clipboard(event.controlWindow()->handle());
 }
 }
}
```



```

// Find the cursored object in the container.
Department* cursoredObject = (Department*)(container().cursoredObject());
// If a Copy request, utilize delayed rendering to put the data of
// of the private format on the clipboard. If a Cut request, put
// the data directly on the clipboard since the object won't be
// around later when the renderFormat code needs it.
// We maintain an "objectList" collection to keep track of
// the objects copied onto the clipboard so that renderFormat
// knows which objects to render.
// We also store text data for the objects so that applications
// that don't support the private format can paste the data into a
// text editor.
// Clear the collection used to keep track of clipboard objects.
objectList->removeAll();
// If the cursored object is selected, loop through all other
// selected objects and store the objects in our set.
if (container().isSelected(cursoredObject))
{
 unsigned long count = 0;
 IString objectsAsText("\0");
 IContainerControl::ObjectCursor cursor(container(), IContainerObject::selected);
 for (cursor.setToFirst(); cursor.isValid(); cursor.setToNext())
 {
 count++;
 Department* selectedObject = (Department*)(container().objectAt(cursor));
 objectList->add(selectedObject);
 objectsAsText = objectsAsText + selectedObject->text();
 }
 // Empty the clipboard to establish ownership
 clipboard.empty();
 // Put the data on the clipboard. We put our private
 // format first since it has the most information.
 // We use 0 for the data on a Copy request because
 // we want to delay the rendering until
 // the paste operation.
 // If this is a Cut, put the data on the clipboard
 // instead of using delayed rendering because we delete
 // the object.
 if (event.commandId() == MI_CUT)
 {
 IString string = selectedData();
 clipboard.setData(Department::renderedFormat, (const char*)string, string.length()+1);
 objectList->removeAll();
 container().deleteSelectedObjects();
 }
 else
 clipboard.setData(Department::renderedFormat, 0, 0);
 clipboard.setText(objectsAsText);
}
else
{
 // If the object is not selected, repeat the above procedure
 // for the cursored object.
 objectList->add(cursoredObject);
 // Empty the clipboard to establish ownership
 clipboard.empty();
 // Put the data on the clipboard (using
 // delayed rendering for a Copy. Delete the object
 // if a Cut request.
 IString objectAsText = cursoredObject->text();
 if (event.commandId() == MI_CUT)
 {
 IString string = selectedData();
 clipboard.setData(Department::renderedFormat, (const char*)string, string.length()+1);
 objectList->removeAll();
 container().removeObject(cursoredObject);
 delete cursoredObject;
 }
}

```

```

 }
 else
 clipboard.setData(Department::renderedFormat, 0,0);
 clipboard.setText(objectAsText);
 }
 return true;
}
case MI_PASTE :
{
 IClipboard clipboard(event.controlWindow()->handle());
 // If the clipboard has data of the private format,
 // get the data and build objects from it.
 // Note: To see the text format of the data, paste
 // from the clipboard using any text editor that
 // supports the clipboard.
 if (clipboard.hasData(Department::renderedFormat))
 {
 IString strCount, strObject, objectsAsString;
 // Query the data on the clipboard.
 char* data = (char*)clipboard.data(Department::renderedFormat);
 // Parse the string into our data fields.
 data >> strCount >> separator >> objectsAsString;
 // Extract the number of objects stored in the string.
 unsigned long count = strCount.asUnsigned();
 // Turn refresh off to eliminate multiple painting.
 container().setRefreshOff();
 // Construct new objects from the data.
 for(int i=0; i< strObject >> separator >> objectsAsString;
 Department* department = new Department();
 department->initializeFromString(strObject);
 container().addObject(department);
 }
 // Enable refresh and refresh the container.
 container().setRefreshOn();
 container().refresh();
 }
 return true;
}
}
return false;
}
}

```

##### 5. Create and display the popup menu.

```

/*-----
ContainerCutPasteHandler::makePopupMenu
Create a pop-up menu with the clipboard actions.
-----*/
bool ContainerCutPasteHandler::makePopupMenu(IMenuEvent& event)
{
 IPopupMenu* popupMenu = new IPopupMenu(CNR_POPUPMENU,
 event.controlWindow());
 // Enable the allowable menu items in the pop-up menu.
 if (popupMenuObject()) {
 ((IContainerControl*)event.controlWindow()->setCursor(popupMenuObject());
 popupMenu->disableItem(MI_PASTE);
 }
 else
 {
 popupMenu->disableItem(MI_CUT);
 popupMenu->disableItem(MI_COPY);
 }
 // Show the menu.
 popupMenu->setAutoDeleteObject();
 popupMenu->show(event.mousePosition());
 return true;
}

```

6. Iterate our collection of clipboard objects and put the data onto the clipboard during the processing of the *renderFormat* function.

```

/*-----
| ContainerCutPasteHandler::renderFormat
| Put our private format data on the clipboard.
|-----*/
bool ContainerCutPasteHandler::renderFormat(IEvent& event,
 const IString& format)
{
 // Use the handler's collection to find the Department objects
 // whose data we need to put on the clipboard.
 IClipboard clipboard(event.controlWindow()->handle());
 if (format == Department::renderedFormat)
 {
 IString objectsAsString = selectedData();
 setDelayedData(event,
 Department::renderedFormat,
 (const char*)objectsAsString,
 objectsAsString.size());
 return true;
 }
 return false;
}

```

7. Initialize Department objects from our string format in the function *initializeFromString* and build the string format for an object in the function *asString*.

```

/*-----
| Department::initializeFromString
| Set the fields of the object by parsing the passed string.
|-----*/
Department& Department::initializeFromString (const IString& renderedString)
{
 IString strName, strAddress;
 renderedString >> strName >> separator >> strAddress;
 setName (strName);
 setAddress(strAddress);
 return *this;
}
// ...
/*-----
| Department::asString
| Render the Department object as a String that we can later use to
| reconstruct a Department object.
|-----*/
IString Department::asString () const
{
 IString strObject = name() + separator + address();
 return strObject;
}

```

#### RELATED CONCEPTS

- “Chapter 20. Clipboard” on page 285
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- Clipboard Sample
- “Add Clipboard Support” on page 286
- “Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

**RELATED REFERENCES**

IClipboard

IClipboardHandler

ICnrMenuHandler

ICommandHandler

IContainerObject

---






## Chapter 21. Resources

*Resources* are user-interface components, such as text strings, icons, bitmaps, and keyboard accelerator tables. You define resources in a *resource script file*, a text file that contains statements describing your resources. For example, you may define a menu, string table, or dialog template in a resource script file. A benefit of defining resources in a resource script file is that you can make changes to resource definitions without affecting the application code itself. This makes supporting multiple languages (for example, English and French) easier because a translator does not have to edit your source code files looking for text to change. After creating a resource script file, you compile it into a *binary-format resource file*, then incorporate the compiled resource file into your application.

You can code a resource script file yourself using the *resource tag language*. See the page on Resource Script Files under the Tools and Utilities category in the reference for more information about this language.

You can use the Integrated Resource Editor (IRE) to graphically edit resources and create dialog boxes.

Use one of the following tools to convert resource files across platforms.

| Tool        | Description                                                                                                                                                                                                                                                                                                                                      |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IRE         | <br><br>Lets you imports a binary or script resource file (or extract the binary resources from an executable or DLL), then save it in either an OS/2 or Windows format. |
| convrc.exe  | <br><br>Converts resource script files between Windows and OS/2.                                                                                                         |
| ircconv.exe | <br>Converts resource script files between Windows and OS/2 on the AIX platform. Motif's resource compiler understands the OS/2 format for resource script files and graphic files.                                                                          |

| Tool        | Description                                                                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ibmpcnv.exe | <p>WIN</p> <p>OS/2</p> <p>Convert bitmap (.bmp), icon (.ico), pointer (.ptr) or cursor (.cur) images between OS/2 readable formats and Windows formats.</p> |

Using the *resource compiler*, you compile your resource script file (a text file with the extension .rc) into a binary-format resource file (a file with the extension .res). You may incorporate the compiled resource file into your application as follows:

- Bind the binary-format resource file to the application's main executable file.
- 

WIN

OS/2

Bind the compiled version of the resources into a separate *dynamic link library* (DLL). You may use this DLL for one or more applications.

- 

AIX

Link the compiled version of the resources in a shared library.

OS/2

AIX

OS/2 and AIX will let you bind only one resource file to a particular executable file. (OS/2 will let you bind only one resource file to a DLL as well.)

WIN

Windows lets you bind multiple resources to an executable.

You can also provide national language versions by storing the resources for each language in a separate resource file. You can then build your application as separate executable versions for each language (each with a different resource file bound to it) or as a single executable with a separate DLL for each language.

### The IBM Open Class Resource Library

The IBM Open Class Resource Library includes the following standardized resources:

- bitmaps (for use as toolbar buttons)
- button text and short and long help text for the provided bitmaps
- container expand/collapse bitmaps
- multimedia bitmaps
- strings for buttons on common dialogs and message boxes

See the topic "Create Toolbar Buttons" on page 252 for a description of bitmaps and button text shipped with VisualAge C++.

The library is cpporr50.dll for Windows and OS/2, and libvacocres.o for AIX.

When building or packaging applications that use these bitmaps or strings, you must provide the IBM Open Class Resource Library. See the topic Package and Distribute an IBM Open Class Application for more information.

### Dialog Templates

A *dialog template* is a data structure that describes a dialog window and its control windows. Windows and OS/2 uses the data in the dialog template to create the dialog window and control windows. An application can create a dialog template at run time, or it can use the system resource compiler to create a dialog-template resource.

#### ► AIX

Motif does not support dialog templates. If you write portable applications, use canvases instead of dialog templates. The Hello World 4 sample application shows you how to do this. You can find this sample in the `samples/ioc/hello4` directory.

#### WIN

#### ► OS/2

Only the IRE will let you convert dialog templates. Neither `convrc.exe` nor `ircnvc.exe` can convert these files.

### Loading Resource Libraries

A *resource library* is the file, such as an executable, dynamic link library (Windows and OS/2) or shared library (AIX), with a compiled resource file bound to it. The `ICurrentApplication` class represents the program that is currently running. Use the following member functions of this class to load and access the resource library of your application:

| Member function    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| setResourceLibrary | <p>Sets the default resource library for Open Class Library resources. Open Class Library uses this resource library to load the resources it provides, such as bitmaps and text for standard toolbar buttons. You specify a resource library using the name of a dynamic link library or a value of 0. If you specify 0, Open Class Library attempts to load its resources from the application's executable file.</p> <p>► AIX</p> <p>If you do not call this function, Open Class Library loads its resources from <code>libvacocres.o</code>.</p> <p>WIN</p> <p>► OS/2</p> <p>If you do not call this function, Open Class Library loads its resources from <code>cpporr50.dll</code>.</p> |

| Member function        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| resourceLibrary        | <p>Obtains a reference to the Open Class Library's resource library. Open Class Library loads the resources that it requires from this resource library. To change the name of the resource library, call <code>ICurrentApplication::setResourceLibrary</code> or set the environment variable <code>ICLUI_RESLIB</code> with the name of the dynamic link library or shared library.</p> <p>If you do not call <code>ICurrentApplication::setResourceLibrary</code> to identify Open Class Library's resource library, <code>ICurrentApplication::resourceLibrary</code> first searches the <code>NLSPATH</code> and then it searches the <code>LIBPATH</code> to locate the library.</p> <p><b>AIX</b></p> <p>If you do not change the name of the resource library, this function returns a handle to <code>libvacocres.o</code>. If this function cannot find the resource library using <code>NLSPATH</code> and <code>LIBPATH</code>, it searches for the resource library in <code>/usr/vacpp/loc/C</code>. You can set the environment variable, <code>ICLUI_RESLIB</code>, using the statement <code>export ICLUI_RESLIB=newname</code> from an operating system command line.</p> <p><b>WIN</b></p> <p><b>OS/2</b></p> <p>If you do not change the name of the resource library, this function returns a handle to <code>cpporr50.dll</code>. You can set the environment variable, <code>ICLUI_RESLIB</code>, using the statement <code>set ICLUI_RESLIB=newname</code> from an operating system command line.</p> |
| setUserResourceLibrary | <p>Sets the default resource library for application-defined resources. Open Class Library uses this resource library to load an application-defined resource when you do not otherwise identify a resource library. You can specify a resource library when constructing an <code>IResourceId</code> object to identify a resource. If you do not call <code>setUserResourceLibrary</code>, the default application-defined resource library is the executable file.</p> <p>You can use an argument of 0 for the <code>resLibName</code> parameter to reset the resource library for application-defined resources to the executable file.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |



| Member function     | Description                                                                                                                                                                                                                                                                                        |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userResourceLibrary | Obtains a reference to the default resource library that Open Class Library uses to load application-defined resources for this application. If you do not explicitly specify an application-defined resource library with a call to setUserResourceLibrary, IResourceId uses the executable file. |

► AIX

You can optionally specify the .so extension when naming a shared library.

► WIN

► OS/2

To specify a DLL as the resource library, do not include its file extension when passing its name to the function.

### Loading Specific Resources

The IResourceLibrary class provides member functions that load resources, such as strings, menus, and bitmaps, from an application's executable file. Use the IDynamicLinkLibrary class to load resources that are stored in a dynamic link library.

Typically, you do not need to use the IResourceLibrary as most classes in Open Class Library that require resources provide functions that accept an IResourceId and load the resources from the resource file themselves.

To obtain either the Open Class Library's resource library or the default user resource library, call ICurrentApplication::resourceLibrary or ICurrentApplication::userResourceLibrary, respectively.

### Searching for Resource Files

You may specify how your application searches for resources libraries with an argument in ICurrentApplication::setResourceLibrary and ICurrentApplication::setUserResourceLibrary. The enumerated type IDynamicLinkLibrary::ESearchLocation defines the following search methods:

| Search Method    | Description                                                                                                                                                            |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| kOSDefaultSearch | Finds a resource library using the PATH or LIBPATH environment variables following the search rules of the native operating system. This is the default search method. |

| Search Method  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| kNLSPathSearch | <p>Finds the dynamic link library using the NLSPATH environment variable after substituting %L with the users' current locale.</p> <ul style="list-style-type: none"> <li> <p>▶ WIN</p> <p>▶ OS/2</p> <p>On these platforms, it is the LANG environment variable.</p> </li> <li> <p>▶ AIX</p> <p>Instead of using the LANG environment, the user's current locale is retrieved from a call to setlocale(LC_MESSAGES, NULL).</p> </li> </ul> <p>Loading a resource library from NLSPATH allows you to have multiple languages of your resource libraries on a single system.</p> |

▶ AIX

When loading a shared library IDynamicLinkLibrary, these classes will search paths specified in the header section of the executable in addition to the NLSPATH or the LIBPATH environment variables.

**RELATED CONCEPTS**

- “Chapter 1. Windows” on page 1
- “Chapter 4. Menus” on page 55
- “Chapter 3. Events and Event Handlers” on page 19

**RELATED TASKS**

- “Convert Application Resources” on page 395
- “Hello World Version 4: Adding Dialogs and Push Buttons” on page 420
- Package and Distribute an IBM Open Class Application
- “Create Toolbar Buttons” on page 252
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

---

## Chapter 22. Client/Server Applications and Dynamic Data Exchange

WIN

OS/2

*Dynamic data exchange* (DDE) is a client/server protocol for communicating between two applications running on the same machine. *Client/server* implies a relationship between two applications where the *client* sends requests to the *server*. The server handles the requests and provides services to the client application which then consumes those services. The server-provided services can be either of the following:

- Data
- The ability to execute commands on behalf of a client

The server can provide these services to one or more clients, and a client can request services from one or more servers. In addition, an application can be a client and a server because it can request services from a server while providing services to clients.

As a client/server protocol, DDE enables data to be dynamically exchanged, and thus shared, between two applications running on the same machine. Applications are shielded from the operating details and can share data once they agree on the type of data being exchanged. DDE works with applications written for Windows or OS/2 Presentation Manager.

DDE is based on a client/server application model, as follows:

1. The DDE client application requests a conversation with the DDE server application.
2. The DDE server accepts and handles the conversation.
3. The DDE client can now send a variety of predefined requests to the server.
4. The DDE server then sends data, as well as positive and negative acknowledgments, to client applications.
5. The only unsolicited communication a DDE server application can have with a client application is to end a conversation. It cannot send requests to the client application.

From the users' perspective, DDE gives them the ability to import and export data between applications. DDE applications can exchange data on an ongoing basis without the users intervening. From the application developers' perspective, DDE is a client/server-based protocol consisting of a set of messages, macros, and data structures.

Typically, in a multiprocessing operating system, messages deliver requests and replies. The following points are true:

- Resources in one application are protected from being accessed by another application.
- Memory protection schemes create obstacles when applications need to share data.

To counteract these points, an operating system provides *interprocess communication* (IPC), which enables information to be exchanged between processes or threads through semaphores, queues, pipes, and shared memory. For these IPC methods to work, each application must do the following:

- Register the IPC method being used
- Contain specific information on how to access the data

In windowed environments, such as Presentation Manager or Windows, the primary IPC method is to pass messages. In this case, the same memory protection that protects the operating system also protects the data passed in these messages. Thus, for one application to send data to another, it allocates a *shared memory object* and gives it to the receiver.

### How the DDE Protocol Works

The Open Class Library DDE protocol works with the operating system APIs because DDE messages are dispatched like other operating system messages. Because DDE uses object windows to communicate, an application that uses this component needs to process window messages. To do this, either execute `ICurrentThread::processMsgs`, even if your application has no interface components, or call the following in your code:

```
IApplication::current().run();
```

#### RELATED CONCEPTS

“Establishing DDE Conversations”

“The DDE Framework Design” on page 306

Open Class Threading Model

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Add DDE Client Support” on page 307

“Add DDE Server Support” on page 309

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Establishing DDE Conversations

When you use DDE, the client application must first initiate a conversation with the server applications. Conversations are based on a two-tier hierarchy: *application* and *topic*. Application is typically the name of the executable program; the topic is typically the name of a file in a spreadsheet or word-processor application. During the initiation, the server determines if it can satisfy the client’s request. The server does not respond to the client if it cannot satisfy the request. If it can satisfy the request, it creates a new window and includes the handle of this new window in its response. A server responds once for each topic it can support and, for each response, the server provides a unique window handle. This guarantees that each window handle pair identifying a conversation link is unique.

Once a DDE conversation is established, client applications can request services from the server through *DDE transactions*. In DDE, all transactions are one-way because the client always issues the request and a server responds. The only exception to this rule is the end transaction. The client specifies what it wants from the server by providing the data item name of a service.

A conversation continues until one of the participants ends it. The conversations can be ended by either participant at any time during the interchange.

#### DDE Transactions

DDE is transaction-based because all communications between DDE client and server applications occur within the context of a DDE conversation. Once a DDE client application has successfully initiated a conversation with a DDE server application, all subsequent communications are in the form of asynchronous transactions. These transactions are asynchronous in nature, because they are implemented using operating system messages that are posted, not sent, to the other application. This can lead to complications because there is a minimal amount of information provided in the DDE messages and control blocks they pass in shared memory to connect a response to a particular request. As a result of these potential complications in any asynchronous communications, the DDE protocol requires server applications to respond to requests from any one client in the exact order they are received. With the Open Class Library, the DDE framework enforces this synchronization for you on the server side. On the client side, the framework keeps track of all outstanding transactions, ensuring they are responded to in the correct order. It also provides you with information about your original request, along with the response data.

#### Using Applications, Topics, and Items

For the purpose of initiating a DDE conversation, the conversation is uniquely identified by the application name of the DDE server and the name of the *topic*, where a topic is a logical data context. For example, for a word processing or spreadsheet DDE server application, the topic is typically the file name of a document or spreadsheet. For a DDE news server, the topic could be an area of interest. Once a conversation has been initiated, it is uniquely identified by a pair of window handles, provided by the client and server applications. An item is a named data object, which can be passed as part of a transaction within a DDE conversation. An item can be anything from a simple data type, such as an integer or character string, to something as large as a bitmap or data file.

#### Specifying Data Formats

Whenever a DDE application requests or provides data, it must specify the format of the data. With this information, DDE applications know how to package, unpackage, or interpret the data they pass between them. Before a data format is used by a DDE application, the application must ensure the data format has a unique identification by registering its name in the system atom table. The operating system defines a number of constant names for industry-standard data formats, and adds them to the system atom table. The Open Class Library DDE framework verifies that any format specified by your application is defined in the atom table. As a DDE client application, you request data in a format supported by the DDE server application. As a DDE server application, you provide data in useful formats that DDE client applications can process.

The standard format for exchanging text data is the Developer's Toolkit for OS/2 constant, `DDEFMT_TEXT`, which is defined to be #1.

#### **RELATED CONCEPTS**

"Chapter 23. Direct Manipulation" on page 313

"The DDE Framework Design" on page 306

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

#### RELATED TASKS

“Add DDE Client Support” on page 307

“Add DDE Server Support” on page 309

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## The DDE Framework Design

On the DDE server side, `IDDETopicServer` is the primary class, and represents a DDE server for a single topic. An `IDDETopicServer` object can have as many concurrent conversations as it needs. These conversations can be with the same or different applications. The only restriction is they must all be on the same topic.

On the DDE client side, `IDDEClientConversation` is the primary class. An `IDDEClientConversation` object represents a single conversation with a single DDE server application. An `IDDEClientConversation` object can be reused. For example, when you end a conversation with a DDE server application, you can use the `IDDEClientConversation` object to begin a conversation with any DDE server application on any topic.

Most `IDDEClientConversation` and `IDDETopicServer` functions have a default behavior, which you can override on each callback function. For an `IDDETopicServer` object, the callback functions pass requests for information or for the server to carry out an action. The default implementation for most functions is to return *false*, indicating the DDE server does not support this particular DDE transaction type. To support a particular transaction type, you must override the function and provide an appropriate implementation. This typically involves setting information or data into the passed DDE event. The `IDDETopicServer::requestData` function is a pure virtual function so you must derive a class from `IDDETopicServer` to provide an implementation for it. In this way, you can build a DDE server application incrementally, supporting only the DDE transaction types you need for your application.

For an `IDDEClientConversation` object, the callback functions pass you the responses that the DDE server application has sent to your requests. The default implementation for these functions is to do nothing. Because you made the request to the DDE server application, you are interested in the response the server has sent to you. You must override these functions to process the data and information that is sent as part of each response. `IDDETopicServer` and `IDDEClientConversation` objects pass data and information to you in the form of event objects.

`IDDEEvent` is the main event class from which most other DDE event classes are derived. The only exceptions to this are the event classes associated with beginning and ending a conversation; `IDDEBeginEvent`, `IDDEEndEvent`, and `IDDEClientEndEvent`.

#### RELATED CONCEPTS

“Chapter 23. Direct Manipulation” on page 313

“Establishing DDE Conversations” on page 304

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Add DDE Client Support”  
“Add DDE Server Support” on page 309  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

---

## Add DDE Client Support

► WIN

► OS/2

Do the following to provide DDE client support in your Windows or OS/2 application:

1. Derive a class from `IDDEClientConversation` to provide implementations for all of the virtual callback functions in which you are interested. You must provide an implementation for the data pure virtual function.
2. Create an object of the class you derived from `IDDEClientConversation`.
3. Initiate a conversation with a DDE server application on a particular topic.
4. Make requests for services from the DDE server. These transactions, or services, are usually requests for data, requests to accept data from you (poke data), or requests to execute commands on your behalf.
5. Handle the responses to your requests that the DDE server sends you (in the virtual callback functions you have overridden).
6. End the conversation with this DDE server application when you do not need any more services from this particular DDE server.
7. Delete the object of the class derived from `IDDEClientConversation` or reuse the object and start over at item 3.

If you want to have multiple concurrent conversations, repeat the same procedure for each simultaneous conversation you need. Remember, although an `IDDEClientConversation` object represents a single conversation with a single server at any point in time, you can reuse the object for a subsequent conversation, after you have ended the current conversation.

### Request a DDE Conversation

You can begin a DDE conversation in the following ways:

- When you construct an `IDDEClientConversation` .
- When you call `IDDEClientConversation::begin`.

The `begin` function requires you to specify the name of the application you wish to have a conversation with, and the name of the topic you are interested in having a conversation about. For example:

```
bool bStarted = aConversation.begin("Weather Server", "Today's Forecast")
```

The `begin` function returns *true* if the `IDDEClientConversation` object succeeds in initiating a conversation with the requested application on the requested topic. If the `IDDEClientConversation` object is already in conversation on any topic, it will throw an `IInvalidRequest` exception. You can reuse the `IDDEClientConversation` object for a subsequent conversation once you have ended the current conversation.

The `IDDEClientConversation` class provides a second version of the `begin` function. Use this if you have already established a DDE conversation with another application, and you have an alternate method of exchanging the requisite window handles.

### Request Data

To request data from a server application, use the `requestData` function. A server application can respond to a request for data in one of two ways:

- By sending the data item in the requested format to the client application.
- By sending a negative acknowledgment indicating it cannot provide the data item in the requested format.

In both cases, the following points are true:

- One of the `IDDEEvent` derived classes is passed to you using a callback function so you can call `IDDEEvent::item` and `IDDEEvent::format` to determine what data item you requested and in what format it should be rendered.
- If the server application sends the requested data item, the `IDDEClientConversation` object creates an `IDDEDataEvent` object and passes a reference to this event by calling the `data` function. This is the one pure virtual function of `IDDEClientConversation`, which means you must provide an implementation for it. You can call the various functions of the `IDDEDataEvent` object to get detailed information about the event.

### Request a Data Hot Link

The DDE protocol supports ongoing links to data items, called *hot links*. Once you establish a hot link with a DDE server application for a particular data item, the server informs you whenever the value of the data changes. To request a hot link to a data item with a server application, you must use the `beginHotLink` function.

After you have established a data hot link, every time the value of the data item changes in the DDE server application, the following occurs:

1. The server sends you the new value of the data item.
2. The `IDDEClientConversation` object receives the updated data item, creates an `IDDEDataEvent` object, and passes a reference to this event when calling the `data` function.

Note that you can also use notification hot links. Every time the value of the data item changes in the DDE server application, the server sends a notification that the value of the data item has changed. When the `IDDEClientConversation` object receives the notification, it creates an `IDDEClientHotLinkEvent` object. You can then request the data when and if you need it.

### End a Hot Link

Use `IDDEClientConversation::endHotLink` and `IDDEClientConversation::endHotLinks` to end hot links. To end a single hot link, call the `endHotLink` function, specifying the name of the data item and, optionally, the format of the data. For example, if you have multiple active hot links with a DDE server application on the Weather Corp data item, each in a different format, end one of the hot links as follows:

```
aConversation.endHotLink("Weather Corp", SZFMT_CPTXT);
```

Use the `endHotLinks` function to end multiple hot links with one function call. There are two ways you can use this function:

1. To end all of the hot links for all data formats of a particular data item, specify the name of the data item.
2. To end all hot links for all data items and data formats for the `IDDEClientConversation` object, call `endHotLinks` with no parameters.

### Poke Data

The DDE protocol introduces the concept of poking data from a DDE client application to a DDE server application. *Poking data* is essentially requesting a DDE



server application to set a specified data item to a value passed by the client application. The `IDDEClientConversation::pokeData` member function sends a data poke request to the conversing DDE server application.

The `IDDEClientConversation` object throws an `InvalidRequest` exception if it is not currently engaged in a conversation and the server application can respond to this request with either a positive or negative acknowledgment. In either case, the `IDDEClientConversation` object creates an `IDDEAcknowledgePokeEvent` object and passes a reference to this event by calling the `pokeAcknowledged` function.

### **Execute Remote Commands**

The DDE protocol allows DDE client applications to send commands and command strings to DDE server applications for remote execution. Typically, you use this to send macros and commands to word processing or spreadsheet applications that support a command language and support this aspect of the DDE protocol. You can use `IDDEClientConversation::executeCommands` to send commands to DDE server applications.

### **End a Conversation**

To end the current conversation, call the `IDDEClientConversation::end`. The `IDDEClientConversation` object first cleans up and then resets all of its object data to their original values when constructed.

For an example of how to set up a DDE client, refer to the DDE Client Sample in the `samples/ioc/dde/ddeclnt` directory.

### **RELATED CONCEPTS**

“Chapter 22. Client/Server Applications and Dynamic Data Exchange” on page 303  
“Establishing DDE Conversations” on page 304  
“The DDE Framework Design” on page 306  
“Chapter 1. Windows” on page 1  
“Chapter 3. Events and Event Handlers” on page 19

### **RELATED TASKS**

DDE Client Sample  
DDE Server Sample  
“Add DDE Server Support”  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

---

## **Add DDE Server Support**

This page also outlines the following tasks:

- Providing Data
- Supporting Hot Links

**WIN**

**OS/2**

When creating a client/server application with DDE, there is no limit to the number of concurrent conversations in which an `IDDETopicServer` object can participate. You can have multiple `IDDETopicServer` objects supporting the same topic in your application. If you choose to do this, you determine how you want to

split the load among the multiple servers you create to support the same topic. An `IDDETopicServer` object always supports the topic it is constructed for; the topic cannot be changed.

To provide DDE server support in your Windows or OS/2 application, do the following for each DDE topic you want to support in your application:

1. Derive a class from `IDDETopicServer` to provide implementations for all of the virtual callback functions associated with the DDE transactions that you want your DDE topic server to support.  
Note that you must provide an implementation for the `requestData` pure virtual function.
2. Create an object of the class you derived from `IDDETopicServer`.
3. Wait for DDE client applications to initiate conversations with you. Accept as many of the conversations as you want.
4. Wait for transaction requests from the DDE client applications with which you are in conversations. Process these requests in the order they arrive. These requests may consist of requests for data, hot links, requests to execute commands, and requests to accept data from the client application.
5. End one or more of the conversations as needed. You can do this at any point.
6. Delete the object of the class derived from `IDDETopicServer`. The `IDDETopicServer` destructor automatically ends all conversations in which the `IDDETopicServer` derived class is currently engaged.

When an `IDDETopicServer` object receives a request to begin a conversation from a DDE client application, it does some preliminary checking to verify that the request is valid for this topic server before calling to see if you want to accept the conversation request. The `IDDETopicServer` object compares the application name and topic name supplied in the request with the names contained in its object data, ignoring mismatches due to differences in case. If either the application name or topic name does not match, the request to begin a conversation is discarded. The exception to this is a zero length `char*` for either the application name or topic name. The DDE protocol allows DDE client applications to use zero-length character strings as global characters (or wildcards) for application and topic names. This is a way for DDE clients to find all topics supported by a particular application, all applications that support a particular topic, or all active application-topic pairs.

### **Providing Data**

When an `IDDETopicServer` object receives a request for data from a DDE client application, it ensures it has an active conversation with the client. If there is a current conversation with the client application, the `IDDETopicServer` object creates an `IDDERequestDataEvent` object and calls the `requestData` function.

If you want to provide the data, call `IDDERequestDataEvent::setData`. If you want to request an acknowledgment from the client application upon receipt of the data, call `IDDERequestDataEvent::requestAck`.

If you cannot provide the requested data, you must return *false* from `IDDERequestDataEvent::requestData`. You can provide specific information to the client application as to why you are not providing the requested data.

### **Supporting Hot Links**

The DDE protocol supports ongoing links to data items, hot links. Send the updated data item value or notification, depending on the type of hot link, whenever the value of the data item changes. When an `IDDETopicServer` object

receives a request for a hot link from a DDE client application, the first thing it does is ensure it has an active conversation with the client. If there is a current conversation with the client application, the IDDETopicServer object creates an IDDEServerHotLinkEvent object.

Keep track of all of the data items for which you have accepted a hot link. When the value of one of these data items changes, call the IDDETopicServer::hotLinkUpdate member function and pass the name of the changed data item.

#### **RELATED CONCEPTS**

“Chapter 22. Client/Server Applications and Dynamic Data Exchange” on page 303  
“Establishing DDE Conversations” on page 304  
“The DDE Framework Design” on page 306  
“Chapter 1. Windows” on page 1  
“Chapter 3. Events and Event Handlers” on page 19

#### **RELATED TASKS**

“Add DDE Client Support” on page 307  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

#### **RELATED REFERENCES**

IDDE  
IDDEAcknowledgeEvent  
IDDEActiveServer  
IDDEBeginEvent  
IDDEDataEvent  
IDDEEndEvent  
IDDEEvent  
IDDEExecuteEvent  
IDDEPokeEvent



---

## Chapter 23. Direct Manipulation

*Direct manipulation* is a user interface technique that lets a user start application functions by manipulating objects. The user begins an action by moving the mouse pointer over an object and then pressing and holding down the drag mouse button while *dragging* the selected object to a new location. The user then *drops* the object onto the new location by releasing the mouse button. For this reason, direct manipulation is also known as *drag and drop*.

Thus, the user can perform operations directly on objects that appear on the desktop or within an application.

Direct manipulation is not limited to objects in containers, as the object can be a text string in an entry field. Also, users can drag and drop an object onto a new location in the current window, onto another object in a window, or onto a different window.

Direct manipulation is used to move, copy, and link objects. Generally, move is the default operation and is accomplished through the use of the mouse button defined for drag and drop. Other operations can be specified with the additional use of augmentation keys: the Ctrl key or a combination of the Shift and Ctrl keys.

To initiate a move operation, the user presses and holds the Shift key while moving the drag mouse button. The visual indication of this operation for OS/2 and Motif is the drag image. For Windows, the visual indication for move is the halftoning of the object.

To initiate a copy, the user presses and holds down the Ctrl key while pressing and moving the drag mouse button. The visual indication of this operation for OS/2 and Motif is the halftoning of the drag image. For Windows, the visual indication for copy is the halftoning of the object with the plus sign displayed in the lower-right corner.

Likewise, users can accomplish a link operation by pressing and moving the drag mouse button while pressing the Shift and Ctrl keys. The visual indication of this operation in OS/2 is a line that is drawn that connects the drag image with the object where the drag was initiated. For Windows, the visual indication is the halftoning of the object with the short cut arrow displayed in the lower-right corner. For AIX, the visual indication is the halftoning of the drag image with the link arrow displayed in the lower-right hand corner.

Users can request help during a direct manipulation operation by pressing the help key F1. This displays help for the object a user is dragging over.

To cancel a direct manipulation operation, press the Esc key.

The Open Class Library provides four main classifications of objects to support direct manipulation:

- A drag item (IDMItem)
- A drag item provider (IDMItemProvider)
- A renderer (IDMSourceRenderer or IDMTargetRenderer)
- An event handler (IDMSourceHandler or IDMTargetHandler)

The collaboration of these objects allows the rendering from a source to a target location. *Rendering* is the process by which data is transferred from the source of a direct manipulation operation to a target. The following is an overview of the process:

1. A user initiates a drag request, which generates an event that is processed by the source handler.

#### WIN

For Windows, this event needs to be generated. Windows leaves it up to the control to decide when a drag operation is occurring. This decision is usually made on a per control basis because Windows allows using the same mouse button, the left button by default, for selection and drag.

`IWindow::isDragStarting` (Windows only) helps to detect a drag operation and is called for controls that have been enabled for drag through `IDMHandler::enableDragFrom` or `IDMHandler::enableDragDropFor`. After a control has been enabled for drag, `IWindow::isDragStarting` is called for every mouse button down, up, or move that is dispatched to the control. When a drag has been detected, the `isDragStarting` override should return true. The return value of true causes `IDMHandler` to send a `WM_BEGINDRAG` message and initiate the IDM classes to start the drag operation.

Once the `WM_BEGINDRAG` message is generated, the IDM classes work the same on all platforms. OS/2 and AIX can generate the `WM_BEGINDRAG` message for you. Windows offers you the option of initiating a drag with the right mouse button for non default drag and drops, which allows differentiating the drag and selection operations. `IWindow` has a default implementation of `isDragStarting` that tracks the left button state along with mouse move messages. If the left button down message is received and the mouse moves more than 2 pels before a left button up event is received, then a drag operation has occurred. If you need to support drag from a control that requires more elaborate drag conditions, override this member function and supply an implementation. You can use `IWindow`, `IEntryField`, `IMultiLineEdit` and `IContainerControl` implementations of `isDragStarting` for examples.

2. The source handler uses the source window's attached drag item provider to request generation of source drag items.

*Drag items* represent the objects that are the focus of the direct manipulation operation and provide access to the object's data. *Drag item providers* are designed to assist in the generation of drag items and bridge the gap between the drag items and the source and target handlers.

When you use the `IDMItemProviderFor` template to instantiate the provider, the static function in the derived drag item class, `generateSourceItems`, is called. Alternatively, if the provider is instantiated from a derived drag item provider class, then the `IDMItemProvider::provideSourceItems` override is called.

3. Once the source drag items are generated, appropriate source renderers are selected.
4. When you enter a potential target window, an event is generated that is processed by the target handler.
5. The target handler uses the target window's attached drag item provider to request generation of target drag items.

When you use the `IDMItemProviderFor` template to instantiate the provider, the target drag item constructor is called, as follows:

```
IDMItem (const Handle& item);
```

Alternatively, if the provider is instantiated from a derived drag item provider class, then the `IDMItemProvider::provideTargetItemFor` override is called.

6. The target handler also uses the drag item provider to request additional verification support via the virtual function, `IDMItemProvider::provideEnterSupport`.
7. Once the target drag items are generated and verified, the appropriate target renderer is selected.
8. The drop is processed based upon the rendering mechanism and format, which is stored in the target renderer, and the data is subsequently transferred. The virtual function, `IDMItem::targetDrop`, is used to process the drop event.

Renderers encapsulate the various mechanisms and formats that are used for rendering and implement the logic that supports the mechanisms and formats (RMFs). Generally, a *mechanism* defines the method of data transfer, whereas the *format* identifies the format of the data. The Open Class Library's implementation groups the mechanisms and formats under one mechanism.

*Target rendering* transfers data from the source to the target. Because the source can package all the required transfer information when the drag operation begins, the target can complete the drop operation without further assistance from the source.

A good example of target rendering involves the use of the Open Class Library's `process RMFTM`. If the source and target are in the same process, the target directly accesses the source's data and subsequently renders the information as required.

On the other hand, *source rendering* occurs when the target requires additional information from the source in order to complete processing of the drop. The target issues the appropriate events to request the information from the source. A good example of source rendering involves the use of the User Interface Class Library's shared memory RMF. When the source and the target are in separate processes, the target can select this RMF that generates a shared memory buffer that, in turn, transfers the data between the source and target.

### Drag Items

Drag items are represented by objects of class `IDMItem`. *Drag items* encapsulate the logic that serves as the bridge between the context-insensitive handlers and renderers and the application-specific behavior of particular source and target windows. Thus, the drag items provide the application-specific semantics of the direct manipulation operation.

`IDMItem` is the base class that defines the general behavior of all direct manipulation items. The three derived classes provided by the Open Class Library provide specializations of the base class that represent the objects being dragged and dropped on specific controls.

The following classes are derived from `IDMItem`:

- `IDMCnrItem`
- `IDMMLEItem`
- `IDMEFIItem`

### Drag Item Providers

The `IDMItemProvider` class is an extension of the `IWindow` class that provides direct manipulation functions. Objects of the `IDMItemProvider` class allow generic controls, such as an entry field, to generate context-sensitive drag items. For

example, a container that contains customer objects can generate a “customer” item; a bitmap can provide an item that can extract the picture from a .BMP file.

The `IDMItemProvider` class also provides functions that deal with target entry and exit, as well as help. With `IDMItemProvider::provideEnterSupport` you can verify objects, using different schemes, when the object is over a potential target.

For example, if you want to restrict drops in an icon view container to the white space area of the container, you can use the `provideEnterSupport` function to determine if the pointing device is over either white space or an actual container object:

```
bool CnrProvider::provideEnterSupport
(IDMTargetEnterEvent &event)
{
 // Allow default verification to occur
 if (Inherited::provideEnterSupport(event))
 {
 // Prevent the drop if we're over a
 // container object (icon view)
 if (event.object())
 {
 event.setDropIndicator(IDM::notOk);
 return(false);
 }
 return(true);
 }
 return(false);
}
```

The return value is not currently used to determine if a drop is allowed. Only the drop indicator is used to set the drop disposition.

#### RELATED CONCEPTS

“Chapter 20. Clipboard” on page 285

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Add Clipboard Support” on page 286

“Use Default Direct Manipulation”

“Add Images to Drag Items” on page 331

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Use Default Direct Manipulation

The Open Class Library provides default direct manipulation support for the following:

- Entry fields
- Multiline edit (MLE) fields
- Containers (within the same process)
- Tool bars (including menu bars and tool bar buttons)

On OS/2, you use the left mouse button to drag objects. On Windows, you use the right mouse button to drag objects. You can use the left mouse button to drag but when the drag operation is finished, you get a pop-up menu requiring a menu



choice of **Copy, Move, or Cancel**. To drag objects on AIX use the middle mouse button or if you have a two button mouse use both buttons simultaneously.

### Using Defaults for Entry Fields and MLEs

The default direct manipulation support for entry fields and MLEs is almost identical. When you create source items, selected text is stored within the item using the `IDMItem::setContentts` function.

In Windows, the text must be selected in order to initiate a drag. In OS/2, all of the text is dragged if none is selected. Afterwards, the optimal rendering mechanisms and formats are determined using the length of the text characters plus any embedded characters. If the text length is fewer than or equal to 255 characters, and does not contain any embedded nulls, the text rendering format, `IDM::rfText`, is used. Otherwise, the shared rendering format, `IDM::rfSharedMem` is used.

To further optimize performance, the process rendering format, `IDM::rfProcess`, is added to the prior selection to handle it when the source and target entry fields or MLEs are located within the same process. If they are located within separate processes, one of the other rendering formats, is used.

The default rendering support for `IDM::rfSharedMem` automatically allocates the shared memory buffer and transfers the data (stored in the source item using `IDMItem::setContentts`) from one process to another when its use is required. Therefore, the data is accessible to `IDMItem::contentts` in the target item after the drop has occurred.

#### WIN

The default drag image style for the entry field and MLE support are the OLE drag cursors.

#### OS/2

The default drag image style for the entry field and MLE support is `IDM::allStacked`.

The default implementation of the `IDMItemProvider::provideEnterSupport` function for the entry field and MLE items prevents a user from dropping text within the same window. The source and target window cannot be the same window.

When the entry field and MLE items differ, default target drop processing occurs. The entry field item retrieves the text using the `IDMItem::contentts` function and sets the text into the entry field. The MLE item appends the text to the end of the MLE field.

### Using Defaults for Containers

The default direct manipulation for containers supports moving or copying container objects within the same process. Also, all of the container views are supported. When you construct a source item, the container object is stored within the source item using the `IDMItem::setObject` function. Because a target can directly address a container object in the same process, you can use `IDMItem::object` at the target to access the container object after the drop has occurred. Therefore, the use of `IDMItem::setContentts` and `IDMItem::contentts` is supported if you extend the default support, but they are not used in the default implementation. Finally, the process rendering format, `IDM::rfProcess`, is set. This is the only RMF used for the default support.

If the user selects multiple container objects, a sequence collection is created. The source items are then stored based upon the following order:

1. The object under the mouse pointer is stored first.
2. The other objects are stored in the order in which `IContainerControl::ObjectCursor` returns them.

The default drag image style for container support is `IDM::allStacked`.

The default implementation of the `IDMItemProvider::provideEnterSupport` function for the container item prevents the user from dropping an object on a target container object if drops have been disabled by the `IContainerObject::disableDrop` function. Also, the `IContainerControl::isMoveValid` function is called to ensure that a move operation, the default, is valid.

Default target-drop processing handles both moving and copying. The default positioning of the dropped items is based upon the view of the target container. If multiple container objects are involved, `IDMCnrItem::targetDrop` is called once for each container item, and the items are processed in the reverse order in which they were added to the sequence collection.

When implementing container copy support, you must define an override for `IContainerObject::objectCopy` in the derived `IContainerObject` class. Also, you must define a copy constructor to be used by the override.

These are illustrated in the `drag3` sample.

### **Enabling Default Support**

You enable the default direct manipulation support for the container, entry field, and MLE by calling the desired `IDMHandler` static function. A pointer to the control object is supplied as the functional parameter to the static function, which performs all the required setup to enable the default support. Two samples, `drag1` and `drag3`, are supplied to illustrate enabling default support.

Containers and windows that support direct manipulation can be the source, target, or source and target of a drag operation. This is determined by the use of the static functions `IDMHandler::enableDragFrom`, `IDMHandler::enableDropOn`, or `IDMHandler::enableDragDropFor`, respectively. However, notice the differences in the support for the menu bar, tool bar, and tool bar buttons.

### **Using Defaults for Tool Bars**

The default direct manipulation for tool bars supports the dropping of menu item objects from a menu bar within the same process to create a new tool bar button. Also, moving tool bar buttons within the same process is supported: you can move and arrange tool bar buttons within the same tool bar or you can move them from another tool bar.

Deleting tool bar buttons is also supported, as you can drop the buttons on a shredder object.

On OS/2, when a source item is constructed from a menu item, the menu item resource identifier is stored within the source item using the `IDMItem::setObject` function, and the menu item text is stored using the `IDMItem::setContent` function. The process rendering format, `IDM::rfProcess`, is set as it is the only RMF used for the default support.

Afterwards, the `IDMMenuItem` constructor attempts to set the drag image based upon a stored image referenced by the resource identifier. The image can be one of the supplied Open Class Library defaults or the user can define it. If one is unavailable, a default image is used. The default drag image style for menu bar support is `IDM::allStacked`. The default operation is `IDMOperation::link`, and the drag item type is `IDM::menuItem`.

The Open Class Library for Windows does not support the dropping of menu item objects from a menu bar to create a new tool bar button.

Because the menu bar is only supported from a source perspective, the `IDMItemProvider::provideEnterSupport` and `IDMItem::targetDrop` functions are not implemented.

When a source item is constructed from a tool bar button, a pointer to the button window is stored within the source item. The process rendering format, `IDM::rfProcess`, is set, as well the shredder RMF, `IDM::rmDiscard`, and `IDM::rfUnknown`. Then, the `IDMTBarButtonItem` constructor sets the drag image based upon the button's stored image. The image can be one of the supplied IBM Open Class Library defaults or the user can define it. If the image is unavailable, a default image is used. The default drag image style for tool bar button support is `IDM::allStacked`. The default operation is `IDMOperation::move`, and the drag item type is `IDM::toolBarButton`.

The default implementation of the `IDMItemProvider::provideEnterSupport` function for the tool bar button item prevents the user from dropping a button on itself. It also filters the drag item types to allow drops only for the following types: `IDM::toolBarButton`, `IDM::menuItem` (OS/2 only), and `IDM::bitmap`. `IDM::bitmap` is included to allow system bitmaps to be dropped on a button. The special case of a system bitmap with a type of `IDM::plainText` is also handled.

Default tool bar button drop processing handles both moving and linking. The default positioning of the dropped item is based upon the position of the object over the tool bar button when it was added to the tool bar. If the new button is dropped on the left half of a tool bar button, the button is moved before the button where the drop occurred. If the new button is dropped on the right half or at the center of the tool bar button, the button is moved after the button where the drop occurred. This rule applies to every source of a drag operation, including tool bar buttons created from a menu bar, buttons within the same tool bar, and buttons from another tool bar. If the tool bar is vertical, a similar rule applies. If the new button is dropped on the lower half of a tool bar button, the new tool bar button is moved below the button where the drop occurred. If the new button is dropped on the upper half, or at the center of the tool bar button, the button is moved above the tool bar button where the drop occurred. Finally, if the item that was dropped was a system bitmap, the current tool bar button image is replaced using the system bitmap.

The tool bar only supports drop processing. The default positioning adds the tool bar button to the end of the tool bar and places it within its own group. If the source of the drag was a menu bar, a new button is created and added to the end of the tool bar. If the source of the drag was a tool bar button within the same tool bar, the button is moved to the end of the tool bar. When the source of the drag was a tool bar button in another tool bar, the button is removed from the source tool bar, and added to the end of the target tool bar.

Tool bar support only works upon menu bars, tool bar buttons, and tool bars that are within the same process.

#### RELATED CONCEPTS

- “Chapter 23. Direct Manipulation” on page 313
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Use Rendering Mechanisms and Formats”
- “Use Drag Item Types” on page 323
- “Enable Direct Manipulation for an Entry Field or MLE” on page 324
- “Enable Direct Manipulation for a Container” on page 325
- “Set and Query the Drag Operation” on page 330
- “Set the Target Emphasis” on page 331
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

## Use Rendering Mechanisms and Formats

*Rendering* is the process by which data is transferred from the source of a direct manipulation operation to the target. If the source and target objects are within the same process, both objects have access to the same memory address space, and the target can readily access the source data to complete the transfer. If the source and target are in separate processes, the data transfer is facilitated using a global (or shared) memory buffer and an operation that involves the dispatching and processing of rendering messages.

*Renderers* transfer the representation of the object being manipulated from the source object to the target object. Direct manipulation renderers manage and maintain rendering mechanisms and formats (RMFs) whose characteristics are defined by the RMF pairs that represent the data transfer method. The rendering mechanisms and formats identify the set of protocols that your items support. These renderers are objects of classes `IDMSourceRenderer` and `IDMTargetRenderer` and are derived from `IDMRenderer`.

When you create an `IDMSourceHandler` object, the Open Class Library creates a default `IDMSourceRenderer`. The following table displays the source RMF pairs and the corresponding drag item type. `IDM::any` represents any drag item type. Any object that you manipulate must have an explicit attribute that identifies the type of the item. These objects are often passed by Presentation Manager mechanisms that need to identify the attributes of an item.

#### Default Source Renderer

| Rendering Mechanism         | Rendering Format              | Item Type             |
|-----------------------------|-------------------------------|-----------------------|
| <code>IDM::rmLibrary</code> | <code>IDM::rfProcess</code>   | <code>IDM::any</code> |
| <code>IDM::rmLibrary</code> | <code>IDM::rfText</code>      | <code>IDM::any</code> |
| <code>IDM::rmLibrary</code> | <code>IDM::rfSharedMem</code> | <code>IDM::any</code> |
| <code>IDM::rmPrint</code>   | <code>IDM::rfUnknown</code>   | <code>IDM::any</code> |
| <code>IDM::rmDiscard</code> | <code>IDM::rfUnknown</code>   | <code>IDM::any</code> |
| <code>IDM::rmFile</code>    | <code>IDM::rfUnknown</code>   | <code>IDM::any</code> |
| <code>IDM::rmObject</code>  | <code>IDM::rfObject</code>    | <code>IDM::any</code> |

When an `IDMTargetHandler` object is created, the Open Class Library creates a default `IDMTargetRenderer`. The default target renderer RMF pairs are shown in the table below.

### Default Target Renderer

| Rendering Mechanism         | Rendering Format              | Item Type             |
|-----------------------------|-------------------------------|-----------------------|
| <code>IDM::rmLibrary</code> | <code>IDM::rfProcess</code>   | <code>IDM::any</code> |
| <code>IDM::rmLibrary</code> | <code>IDM::rfText</code>      | <code>IDM::any</code> |
| <code>IDM::rmLibrary</code> | <code>IDM::rfSharedMem</code> | <code>IDM::any</code> |
| <code>IDM::rmFile</code>    | <code>IDM::rfUnknown</code>   | <code>IDM::any</code> |
| <code>IDM::rmObject</code>  | <code>IDM::rfObject</code>    | <code>IDM::any</code> |

The Open Class Library provides `IDM::rmLibrary` as the rendering mechanism used for efficient drag and drop operations. The following table displays other rendering messages defined as part of the default renderers.

### Other Default Rendering Mechanisms

| Rendering Mechanism                     | Used When...                                                                                                                                                 |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>IDM::rmPrint</code> (OS/2 only)   | An Open Class Library object is dropped on a printer                                                                                                         |
| <code>IDM::rmDiscard</code> (OS/2 only) | An Open Class Library object is dropped on the shredder                                                                                                      |
| <code>IDM::rmFile</code> (OS/2 only)    | A file is dragged from the source and dragged over or dropped on a target                                                                                    |
| <code>IDM::rmObject</code> (OS/2 only)  | A Workplace Shell <sup>®</sup> object is processed. Your application may be required to run under a Workplace Shell process to use this rendering mechanism. |

Several default rendering formats are defined to assist you in using the direct manipulation classes. The following table displays these default rendering formats.

### Default Rendering Formats

| Format                        | Used When...                                                                                                                                                                                                          |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>IDM::rfProcess</code>   | Determining if the source of the direct manipulation operation and the target are in the same process. This format must be constructed by calling the static member function <code>IDMItem::rfForThisProcess</code> . |
| <code>IDM::rfText</code>      | Dragging text that has a length of 255 or fewer characters with no embedded null characters.                                                                                                                          |
| <code>IDM::rfSharedMem</code> | A shared memory buffer is required to transfer the data from the source to the target. This format should be used when transferring data between two separate processes and <code>IDM::rfText</code> cannot be used.  |

| Format                    | Used When...                                                                                                                                 |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| IDM::rfUnknown            | The format is unknown.                                                                                                                       |
| IDM::rfObject (OS/2 only) | A Workplace Shell object is processed. Your application may be required to run under a Workplace Shell process to use this rendering format. |

You can use the `IDM::any` type to represent any drag item type.

The *native* renderer is the first rendering mechanism and format defined when you create the item. For example, in the declaration of the default source renderer, the native renderer supports the library rendering mechanism, the process rendering format, and any item type. In the declaration of the default target renderer, the native renderer supports the library rendering mechanism, the process rendering format, and any item type.

The following table displays the default Open Class Library RMF pairs that support target rendering.

#### Target RMFs

| Mechanism      | Format         |
|----------------|----------------|
| IDM::rmLibrary | IDM::rfProcess |
| IDM::rmLibrary | IDM::rfText    |
| IDM::rmFile    | IDM::rfUnknown |
| IDM::rmFile    | IDM::rfText    |
| IDM::rmObject  | IDM::rfObject  |

The following table displays the default Open Class Library RMF pairs that support source rendering.

#### Source RMFs

| Mechanism      | Format           |
|----------------|------------------|
| IDM::rmLibrary | IDM::rfSharedMem |
| IDM::rmPrint   | IDM::rfUnknown   |
| IDM::rmDiscard | IDM::rfUnknown   |

To create renderers for controls not supported by the Open Class Library, you can create your own source or target renderer. To do this, derive from the `IDMSourceRenderer` or `IDMTargetRenderer`, create instances, and then add them to the handler using `setDefaultTargetRenderer` and `setDefaultSourceRenderer`.

#### RELATED CONCEPTS

- “Chapter 23. Direct Manipulation” on page 313
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Use Default Direct Manipulation” on page 316  
“Use Drag Item Types”  
“Enable Direct Manipulation for an Entry Field or MLE” on page 324  
“Enable Direct Manipulation for a Container” on page 325  
“Set and Query the Drag Operation” on page 330  
“Set the Target Emphasis” on page 331  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

## Use Drag Item Types

Drag item types are useful in distinguishing drag items. Normally, the type is defined when the drag item object is constructed. `IDMItem` functions, such as `IDMItem::setTypes` and `IDMItem::types`, are defined to allow the setting and querying of the types, respectively.

The Open Class Library defines the following default types that you can use in your application:

**IDM::any**  
Any type

**IDM::binary**  
Generic binary item type

**IDM::binaryData**  
Binary data item type

**IDM::bitmap**  
Bitmap item type

**IDM::container**  
Container item type

**IDM::containerObject**  
Container object item type

**IDM::file**  
File item type

**IDM::icon**  
Icon item type

**IDM::menuItem**  
Menu item drag item type (OS/2 only)

**IDM::plainText**  
Plain text drag item type

**IDM::text**  
Generic text drag item type

**IDM::toolBarButton**  
Tool bar button drag item type

**IDM::unknown**  
Unknown drag item type

You can define new types as required by your application if the preceding list does not have the types you need.

#### RELATED CONCEPTS

- “Chapter 23. Direct Manipulation” on page 313
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Use Default Direct Manipulation” on page 316
- “Use Rendering Mechanisms and Formats” on page 320
- “Enable Direct Manipulation for an Entry Field or MLE”
- “Enable Direct Manipulation for a Container” on page 325
- “Set and Query the Drag Operation” on page 330
- “Set the Target Emphasis” on page 331
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

## Enable Direct Manipulation for an Entry Field or MLE

The following sample shows you how to enable direct manipulation for an entry field or an MLE control and how to use the static function, `IDMHandler::enableDragDropFor`. This static function creates the following:

- Source and target handlers
- Source and target default renderers
- An entry field item provider

In the following, from the Direct Manipulation Sample 1, the highlighted lines enable direct manipulation of text between two entry fields in the same process. Direct manipulation is enabled the same way for an MLE. The complete sample is located in the `samples/ioc/drag1` directory.

```
#include <iframe.hpp>
#include <ientryfd.hpp>
#include <idmefit.hpp>
#include <idmhndlr.hpp>
#include "dmsamp1.h"
/*****
* main
* - Application entry point.
*****/
int main()
{
 // Create a generic frame window.
 IFrameWindow frame(WND_MAIN);
 // Create 2 entry fields for the client area.
 IEntryField
 client(1000, &frame, &frame),
 ext (1001, &frame, &frame);
 // Enable source and target direct manipulation support
 // for both entry fields.
 IDMHandler::enableDragDropFor(&client);
 IDMHandler::enableDragDropFor(&ext);
 // Frame setup - Put both entry fields in the client
 // area, with one added as an extension.
 frame
 .setIcon(frame.id())
 .setClient(&client)
 .addExtension(&ext, IFrameWindow::belowClient, 0.5)
 .setFocus()
 .show();
}
```



```

// Run Direct Manipulation Sample 1
IApplication::current().run();
return 0;
}

```

The preceding sample illustrates how you can enable direct manipulation if you only need default entry field support. If you substitute `IMultiLineEdit` for `IEntryField` when constructing `client` and `ext`, the preceding sample then demonstrates the default MLE support.

You can find the Direct Manipulation Sample 1 in the `samples/ioc/drag1` directory.

#### RELATED CONCEPTS

“Chapter 23. Direct Manipulation” on page 313  
 “Chapter 1. Windows” on page 1  
 “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

Direct Manipulation Sample 1  
 “Use Default Direct Manipulation” on page 316  
 “Use Rendering Mechanisms and Formats” on page 320  
 “Use Drag Item Types” on page 323  
 “Enable Direct Manipulation for a Container”  
 “Set and Query the Drag Operation” on page 330  
 “Set the Target Emphasis” on page 331  
 “Write an Event Handler” on page 25  
 “Task and Samples Cross-Reference Table” on page 445

## Enable Direct Manipulation for a Container

This section shows you how to enable direct manipulation for a container and how to use the `IDMHandler` static functions `enableDragFrom` and `enableDropOn`.

In the following, from the Direct Manipulation Sample 3, the `dmsamp3.hpp` file defines a container control object. The `dmsamp3.cpp` file creates the container and container objects and, in the highlighted lines, calls `IDMHandler::enableDragFrom` and `IDMHandler::enableDropOn`. You can find the following sample in the `samples/ioc/drag3` directory:

```

int main()
{
 MySourceWin sourceWin(WND_SOURCE);
 MyTargetWin targetWin(WND_TARGET);
 IApplication::current().run();
 return 0;
}
// ...
MySourceWin :: MySourceWin(unsigned long windowId)
 : MyWindow(windowId)
{
 ITitle title(this, TITLE_SOURCE);
 // Enable the source for dragging from (only).
 IDMHandler::enableDragFrom(cnrCtl);
};
/*****
* MyTargetWin :: MyTargetWin *
* - Constructor *
*****/
MyTargetWin :: MyTargetWin (unsigned long windowId) :
 MyWindow (windowId)

```

```

{
 ITitle title(this, TITLE_TARGET);
 // Enable the target for dropping on (only).
 IDMHandler::enableDropOn(cnrCtl);
}
/*****
* Customer :: Customer
* - Copy Constructor
*****/
Customer :: Customer(const Customer &cnrobj)
 : IContainerObject((const IContainerObject&) cnrobj),
 strName(cnrobj.name()),
 strAddress(cnrobj.address()),
 strPhone(cnrobj.phone()),
 myWin(cnrobj.myWin)
{
}
// ...
/*****
* Customer :: objectCopy
* - Make a copy of the Customer object. Called by
* IContainerObject::copyObjectTo().
*****/
IContainerObject* Customer :: objectCopy()
{
 // Use Customer copy constructor to make a copy of the
 // object.
 Customer *copy = new Customer(*this);
 return((IContainerObject *)copy);
}

```

- The MySourceWin constructor creates a source window. This constructor calls IDMHandler::enableDragFrom. This function enables the window as a drag source.
- The MyTargetWin constructor creates a target window. This constructor calls IDMHandler::enableDropOn. This function enables the window as a drop target.
- The Customer::objectCopy function overrides IContainerObject::objectCopy. This function calls the Customer copy constructor.

The preceding sample illustrates how you can enable direct manipulation if you only need default container support. The Direct Manipulation Sample 3 is located in the samples/ioc/drag3 directory.

The previous container example only illustrates intraprocess (source and target containers are in the same process) container support. The following sample, from the Direct Manipulation Sample 4, shows interprocess (source and target containers are in separate processes) container support. You must start two copies of this sample, drag4, to view the interprocess support.

The module dmsamp4.cpp contains the key logic for the drag4 sample. You can find the following sample in the samples/ioc/drag4 directory:

```

// ...
/*****
| CustomerItem::CustomerItem
|
| Constructor.
|_____*/
CustomerItem :: CustomerItem (const IDMItem::Handle& item) :
 IDMCnrItem (item)
{
 IString
 rmf1 = IDMItem::rmfFrom(IDM::rmLibrary, IDM::rfSharedMem),
 rmf2 = IDMItem::rmfFrom(IDM::rmDiscard, IDM::rfUnknown);
}

```

```

/*****
/* Get pointer to the associated Customer container object */
/*****
Customer *pCustomer = (Customer *)object();
/*****
/* Build and set contents. We can only do this on the source */
/* side. Note that since we call this constructor on both source */
/* and target sides, we must distinguish them. That is done */
/* here by checking the "object" pointer. If this constructor was */
/* called from within our generateSourceItems, then this value */
/* will be non-zero. If called from with the template provider's */
/* provideTargetItemFor, then it will be zero. */
/*****
if (pCustomer)
{
 IString
 contents,
 delim = '\x01';
 contents += pCustomer->iconText() + delim;
 contents += pCustomer->name() + delim;
 contents += pCustomer->address() + delim;
 contents += pCustomer->phone() + delim;
 contents += pCustomer->iconId();
 setContents(contents);
/*****
/* Add RMFs supported by this class (IDMCnrItem will have */
/* already specified the other RMFs we use) */
/*****
 addRMF(rmf1);
 addRMF(rmf2);
}
else
{
/*****
/* On target side, add in <rmLibrary,rfSharedMem> if source concurs */
/* (and it's not already in there)... */
/*****
 if ((item->supportsRMF(rmf1)) &&
 !(supportsRMF(rmf1)))
 {
 addRMF(rmf1);
 }
}
}
*/-----
CustomerItem::generateSourceItems
Called to give CustomerItem opportunity to attach new CustomerItem's to the
argument IDMSourceOperation object.
*/-----
bool CustomerItem :: generateSourceItems (IDMSourceOperation* pSrcOp)
{
/*****
/* Get generic container items. Note that we call the inherited */
/* function since it already has logic to deal with multi-selection, */
/* etc... */
/*****
bool result = Inherited::generateSourceItems(pSrcOp);
/*****
/* Now, replace each IDMCnrItem with a CustomerItem */
/*****
for (unsigned i = 1; i <= pSrcOp->numberOfItems(); i++)
{
 pSrcOp->replaceItem(i, new CustomerItem(pSrcOp->item(i)));
}
/*****
/* Set stack3AndFade as the default image style and set the stacking */

```

```

/* percentage that is used to set the stacking offset as a percentage */
/* of the image size. */
/*****
pSrcOp->setImageStyle(IDM::stack3AndFade);
pSrcOp->setStackingPercentage(IPair(25, 25));
return(result);
}
*/-----
CustomerItem::supportedOperationsFor
Called when a CustomerItem is dropped on a target container.
-----*/
unsigned long CustomerItem ::
 supportedOperationsFor (const IString& rmf) const
{
 if (rmf == IDMItem::rmfFrom(IDM::rmLibrary, IDM::rfSharedMem))
 {
 /*****
 /* If using <rmLibrary,rfSharedMem> then only copy is supported */
 /*****
 return(IDMItem::copyable & supportedOperations());
 }
 /*****
 /* Otherwise, whatever base class supports... */
 /*****
 return(Inherited::supportedOperationsFor(rmf));
 }
}
*/-----
CustomerItem::sourceDiscard
Called when a CustomerItem is dropped on a Workplace Shell shredder.
-----*/
bool CustomerItem :: sourceDiscard (IDMSourceDiscardEvent& event)
{
 /*****
 /* Remove the object from the container */
 /*****
 IContainerControl
 *pCnr = (IContainerControl *) (event.sourceOperation()->sourceWindow());
 IContainerObject
 *pCnrObj = (IContainerObject *) (object());
 pCnr->removeObject(pCnrObj);
 return(true);
}
*/-----
CustomerItem::targetDrop
Called when a CustomerItem is dropped on a target container.
-----*/
bool CustomerItem :: targetDrop (IDMTargetDropEvent& event)
{
 bool result = true;
 /*****
 /* Check if using ICLUI shared memory rendering format */
 /*****
 IString myRMF = IDMItem::rmfFrom(IDM::rmLibrary, IDM::rfSharedMem);
 if (selectedRMF() == myRMF)
 {
 /*****
 /* Yes, construct new Customer object from passed data */
 /*****
 IString
 contents = this->contents(),
 delim = '\x01',
 text = contents.subString(1, contents.indexOf(delim) - 1);
 contents = contents.subString(contents.indexOf(delim) + 1);
 IString

```

```

 name = contents.subString(1, contents.indexOf(delim) - 1);
contents = contents.subString(contents.indexOf(delim) + 1);
IString
 addr = contents.subString(1, contents.indexOf(delim) - 1);
contents = contents.subString(contents.indexOf(delim) + 1);
IString
 phone = contents.subString(1, contents.indexOf(delim) - 1),
 iconId = contents.subString(contents.indexOf(delim) + 1);
IContainerControl *pCnr = event.container();
Customer *pNewCustomer = new Customer(text,
 iconId.asUnsigned(),
 name,
 addr,
 phone,
 (MyWindow *) (pCnr->parent()));
/*****
/* ..and insert the new Customer object into the container */
/*****
pCnr->addObject(pNewCustomer);
/*****
/* Create an IDItem::Handle */
/*
/* Note: We must break this into 2 statements due to a bug in the */
/* IRefCounted class. If we use an initializer to create */
/* the handle, this sample will eventually trap due to the */
/* inability of the initializer to properly increment the */
/* drag item object use count: */
/* IDItem::Handle thisHandle = this; //initializer form */
/*
/* When we break the create into 2 statements, it takes the */
/* form of an assignment which does not have the problem. */
/*****
IDItem::Handle thisHandle;
thisHandle = this;
/*****
/* Position the object within the container */
/*****
IPoint pos = targetOperation()->dropPosition(thisHandle, event);
pCnr->moveObjectTo(pNewCustomer,
 0,
 pCnr,
 0,
 pos);
}
else
{
/*****
/* Some other RMF, base class must support it */
/*****
result = Inherited::targetDrop(event);
}
return(result);
}
// ...

```

The Open Class Library's shared memory rendering format provides the interprocess support. The shared memory format uses a shared memory buffer to transfer the container object data that is stored in the source item (using `IDMItem::setContents`, called in the `CustomerItem` constructor) to the target item where the data can be retrieved (using `IDMItem::contents`, called in `CustomerItem::targetDrop` ). Remember that the source item and target items are in separate processes.

The sourceDiscard function demonstrates container object removal after the user drops the object on the shredder or recycle bin. The supportedOperationsFor function lets you determine which operation or operations a drag item supports based upon the selected rendering mechanism and format. For example, you could make the item IDMItem::copyable, as shown in the preceding example, if the selected RMF is the Open Class Library shared memory RMF. For other RMFs, you could let the drag item default to IDMItem::moveable.

The CustomerItem constructor is generally used to construct target items because it is automatically called for target item construction when using the IDMItemProviderFor template. However, this sample shows how to use it to construct source items as well. The CustomerItem constructor uses the IDMItem::object function to determine if a source or a target item is being constructed. The call to replaceItem in the CustomerItem::generateSourceItems function is the key, as it calls the constructor to create the source item.

The Direct Manipulation Sample 4 is found in samples/ioc/drag4 directory.

#### **RELATED CONCEPTS**

“Chapter 23. Direct Manipulation” on page 313  
“Chapter 1. Windows” on page 1  
“Chapter 3. Events and Event Handlers” on page 19

#### **RELATED TASKS**

Direct Manipulation Sample 3  
Direct Manipulation Sample 4  
“Use Default Direct Manipulation” on page 316  
“Use Rendering Mechanisms and Formats” on page 320  
“Use Drag Item Types” on page 323  
“Enable Direct Manipulation for an Entry Field or MLE” on page 324  
“Set and Query the Drag Operation”  
“Set the Target Emphasis” on page 331  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

## **Set and Query the Drag Operation**

The default operation for direct manipulation is IDMOperation::drag. The direct manipulation target determines the type of operation (for example, move, copy, or link) based upon the allowable operations defined by the item. However, you can override this setting in a derived item’s IDMItem::generateSourceItems function using IDMOperation::setOperation.

The target continually updates this setting, which can be dynamically manipulated using the keyboard augmentation keys. It can be queried using the IDMOperation::operation function.

If the direct manipulation source needs to determine which operation occurred at the target, the operation can be queried using the IDMSourceOperation::operation function. This is sometimes required in a derived item’s IDMItem::sourceEnd function override.

For example, you could distinguish a move from a copy operation so you remove the object from the source if you were performing a move operation

#### RELATED CONCEPTS

- “Chapter 23. Direct Manipulation” on page 313
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Use Default Direct Manipulation” on page 316
- “Use Rendering Mechanisms and Formats” on page 320
- “Use Drag Item Types” on page 323
- “Enable Direct Manipulation for an Entry Field or MLE” on page 324
- “Enable Direct Manipulation for a Container” on page 325
- “Set the Target Emphasis”
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

## Set the Target Emphasis

The `IDMTargetEvent::presSpace` and `IDMTargetEvent::releasePresSpace` functions are defined to assist `IDMTargetEnterEvent`, `IDMTargetLeaveEvent`, and `IDMTargetDropEvent` events in the drawing and removal of target emphasis. You must use these functions to acquire and release the presentation space that is used to draw target emphasis. `IWindow::presSpace` and `IWindow::releasePresSpace` do not work. The `drag2` sample contains a simple implementation of target emphasis support.

#### RELATED CONCEPTS

- “Chapter 23. Direct Manipulation” on page 313
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Use Default Direct Manipulation” on page 316
- “Use Rendering Mechanisms and Formats” on page 320
- “Use Drag Item Types” on page 323
- “Enable Direct Manipulation for an Entry Field or MLE” on page 324
- “Enable Direct Manipulation for a Container” on page 325
- “Set and Query the Drag Operation” on page 330
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

---

## Add Images to Drag Items

When you drag an object, a visual image is displayed for that object. The Open Class Library provides default system images, or you can change the image style and provide your own images.

To change the drag image style, use the `IDMSourceOperation::setImageStyle` member function. We recommend that you call `setImageStyle` from the `IDMItem::generateSourceItems` member function of the application’s derived item class.

The following table describes the `IDMImage` enumerated types and the steps you must take to use them:

| IDMImage Enumeration | Description                                                                                                                                                                                                                                                | What to code                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|
| systemImages         | If one item is dragged, the ISystemPointerHandle::singleFile icon is used. For more than one item, the ISystemPointerHandle::multipleFile icon is used. Any images supplied with drag items are ignored.                                                   | Default                                           |
| allStacked           | Shows each image provided in each drag item. If no images are specified, system images are used.                                                                                                                                                           | Attach IDMImage objects to each IDMItem object    |
| stack3AndFade        | Shows the first three images provided in the drag items and then shows a special icon that looks like the rest of the images fading out. This is optimal when the user can drag more than three items. If no images are specified, system images are used. | Attach IDMImage objects to three IDMItem objects. |

You can use the `IDMSourceOperation::setStackingPercentage` member function to define the stacking percentage used to calculate the placement of the next stacked image when the image style, `IDM::stack3AndFade` or `IDM::allStacked`, is specified. By default, the percentage for both the x- and y- axis is defined as 50 percent of the current image's size, which results in the placement of the origin (bottom left-hand corner for OS/2 applications) of the next image in the center of the current image. The placement of the first image is determined by the position of the mouse pointer. The default direction of stacking is toward the upper right. Increase the stacking percentage to expand the stacking of images, and conversely, decrease the stacking percentage to compress the stacking of images. Also, you can alter the direction of stacking using negative percentages as shown below:

```
IDMSourceOperation::setStackingPercentage(IPair(x, y)); //stacking direction is upper right
IDMSourceOperation::setStackingPercentage(IPair(-x, y)); //stacking direction is upper left
IDMSourceOperation::setStackingPercentage(IPair(-x, -y)); //stacking direction is lower left
IDMSourceOperation::setStackingPercentage(IPair(x, -y)); //stacking direction is lower right
```

Here, x is the stacking percentage for the x-axis and y is the stacking percentage for the y-axis.

You can set this function only once per each drag operation.

The `drag4` sample contains a simple illustration of the use of the `IDMSourceOperation::setStackingPercentage` function.

You can attach `IDMImage` objects to `IDMItem` objects by using the `IDMItem::setImage` member function in the following:

- The constructor of the derived item object
- The implementation of the `IDMItem::generateSourceItems` member function

The following example adds the text I-beam pointer as an image to a derived `IDMItem` in its constructor:



```
MyItem::MyItem (IDMSourceOperation* pIDMSrcOp)
{
 IDMImage image = IDMImage(ISystemPointerHandle(
 ISystemPointerHandle::text));
 setImage(image);
}
```

### **Drag Image Resources for stack3AndFade**

When you use the `IDMItem::stack3AndFade` enumerated type, the Open Class Library uses a fade icon that looks like the images are fading out. This icon is one of many resources that the IBM Open Class provides for you. To package this resource in your application, refer to [Package and Distribute an IBM Open Class Application](#).

#### **RELATED CONCEPTS**

“Chapter 23. Direct Manipulation” on page 313  
IBM Open Class Libraries, Headers and Conventions  
“Chapter 1. Windows” on page 1  
“Chapter 3. Events and Event Handlers” on page 19

#### **RELATED TASKS**

“Use Default Direct Manipulation” on page 316  
[Package and Distribute an IBM Open Class Application](#)  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445



---

## Chapter 24. Multimedia Devices

WIN

OS/2

A multimedia application that integrates text and graphics with a combination of audio, motion video, images, and animation makes your application more attractive to the user, easier to use, and offers better mapping to real world objects.

You can use the window control classes and multimedia classes to implement an interface for your application that looks like the controls of common electronic devices, such as stereo components and video cassette recorders (VCRs). Your application can use these controls as interfaces to control audio and video media that is presented to the user.

Keep in mind user expectations when you create your user interface. Mapping your user interface to the mental image the user has of real-world devices greatly enhances the ease of use of your product. For instance, most users are familiar with the play, stop, pause, fast forward, and reverse controls of an audio cassette recorder.

To use the multimedia classes, ensure that your working environment meets the following requirements:

| Multimedia Task                                           | Hardware Required                    |
|-----------------------------------------------------------|--------------------------------------|
| Using MIDI, audio classes, or software motion video sound | Windows or OS/2-supported sound card |
| Using CDXA or AudioCD                                     | CDROM (OS/2 only)                    |
| Hearing any sounds (CD or otherwise)                      | Speakers or headphones               |
| Recording audio                                           | Microphone                           |

A *medium* is a carrier of information. A *multimedia computer system* is one that is capable of input or output on more than one medium. With the new class of computers, information in virtually any format can be combined into multimedia presentations.

Multiple types of input allow the user to interact with the computer in a style that best suits the information to be communicated, thus relieving overloaded input channels, such as a keyboard, mouse, or microphone.

Output information can be presented in a variety of formats. Typically, output implies a computer display, video, or audio. Video has the potential to hold people's interest and illustrate concepts better than static images. Audio and speech contribute a unique quality to the multimedia system and can increase the information's content.

### Multimedia Device Classes

The Open Class Library supports audio adapters, CD-ROM drives, video-disc players, logical devices, amplifier-mixers, and other hardware devices as media

devices. These media devices are abstracted into classes that contain the data and functions essential for the operation of the real-world devices that they model.

The classes you define for your application combine the capabilities of several classes. Before defining the objects your application needs, choose real-world models that the user knows how to manipulate for the interfaces. You can then use the appropriate Open Class Library multimedia classes that provide the corresponding functions.

The base device classes let you create multimedia devices for your application. The following table lists the base classes and refers to the appropriate sections describing the multimedia devices. In addition, these sections describe how to use the devices.

You can directly instantiate device objects from the following classes:

| Device                          | Class           |
|---------------------------------|-----------------|
| Audio amplifier-mixer           | IMMAmpMixer     |
| CD audio player                 | IMMAudioCD      |
| CD Extended-Architecture player | IMMCDXA         |
| Digital video player            | IMMDigitalVideo |
| Master audio                    | IMMMasterAudio  |
| MIDI sequencer                  | IMMSequencer    |
| Waveform audio player           | IMMWaveAudio    |

### Abstract Device Classes

Common functions for devices are made available through *abstract device classes*. That is, abstract device classes allow inheriting classes to reuse common functions. Note that you cannot instantiate objects from these classes. The following sections describe the multimedia base class and abstract device classes. The multimedia base class (IMMDevice) is the parent class for the family of multimedia classes, including the base device classes and the other abstract classes.

| Device            | Purpose                                                           |
|-------------------|-------------------------------------------------------------------|
| IMMPlayableDevice | Used for many tasks, such as playing, pausing, and seeking.       |
| IMMFileMedia      | Used for devices that work with files.                            |
| IMMRecordable     | Records, saves, cuts, pastes, allows undo, allows redo, saves-as. |
| IMMRemovableMedia | Opens and ejects media; unlocks and locks doors.                  |

### Base Functions for Multimedia

All of the multimedia device classes inherit from the IMMDevice class. This abstract class contains all of the common functions for device objects. These functions include the following:

- Querying the capabilities of a device
- Opening and closing devices
- Changing the speed and time formats
- Changing the audio (on or off)

- Controlling the volume

### **Playable Device**

Objects are usually instantiated from this class in applications that manage different types of devices (such as a VCR and CD remote). You do not create an actual device, rather the instantiated object is used to point to a device a user wants to activate (such as a video player).

An object instantiated from `IMMPlayableDevice` is capable of performing tasks that a home device does to play such media as CDs or video tapes. In addition to the common device functions, such as play, pause, and seek for devices that support playback, `IMMPlayableDevice` objects are able to perform resume, stop, and query position and length operations.

### **WIN**

When using the multimedia classes on the Windows operating system, for audio and video devices you need to load the appropriate file before using any functions on those devices. Otherwise, an exception is thrown.

### **Audio Devices**

Audio input and output is usually in the form of wave or MIDI files.

There is a distinction between sound and music; while it might not be distinctive to a radio, it is to a computer. Sound, such as the sound in wave files, is basically just digitized data that a computer cannot process. Your system plays whatever is in a wave file out to your speakers. By comparison, music, such as the music in MIDI files, is actual information.

### **MIDI Concepts**

MIDI is a standardized set of data blocks or “messages” that instructs any MIDI-compatible sound source as to which notes to play. Rather than representing actual sound recordings, as a file of digitized audio does, a MIDI file merely describes what notes to play and includes settings for the sound or instrument, duration, stereo pan position (how far left or right), and volume.

A MIDI file is comprised of variable-length chunks. There are two types of chunks: a header chunk and one or more track chunks. The number of chunks are defined in the header. A MIDI event can be one of a number of things. It can be a message that turns a particular note on or off, that changes the voice being played by a particular channel, or that defines something about the piece being played.

When you create multimedia applications that play instrumental music, handle it with MIDI (music) rather than digitized audio files (sound). The relative size of the files involved is one of the best arguments for doing so. Compact discs (CDs) are common and hold large amounts of information.

Digital synthesis methods, either FM or wavetable playback, are customarily driven by MIDI.

MIDI files have the extension `.mid` and deliver more music per byte than other formats. MIDI files are comparable in size to ASCII text files, while the other music and sound formats (for example, wave) are comparable to color bitmaps. A digital audio recording of a musical instrument performance can consume large amounts of storage; a MIDI file describing that same performance can take only a kilobyte (K) or two. Wave and CD audio files can sometimes be too big to distribute easily, whereas MIDI files are smaller. Compare 5 minutes of sound in a MIDI file to a 20

MB-wave file. The MIDI file takes about 10K of storage whereas the wave file takes 20 MB. There is a noticeable difference with your application's performance.

MIDI files do not support voice or words. The main role for MIDI in multimedia is music composition and production. Once the music is recorded, it can be played on a high-end synthesizer and recorded in wave or CD-audio formats.

### Waveform Concepts

*Waveform* refers to a digital representation of an original audio sound wave. *Audio* refers to sound waves that have a perceived effect on the human ear.

Digital recordings offer more consistency than MIDI files. A CD recording of music sounds virtually the same on any CD player you use, but a MIDI musical file could sound like, for instance, a French horn on one synthesizer and a kazoo on another. The sound depends on the quality of the sound card. However, MIDI music typically sounds cleaner, more realistic, and more professional than the digital recording, especially if you do not have a sound studio to record your tracks.

Wave files have the extension .wav and contain analog sound that has been recorded digitally. The pieces of sound are usually sampled sound stored as data. An *analog-to-digital converter* creates sampled sound. A wave file can reproduce sound with anything from telephone to compact disc quality in monaural or stereo under computer control.

### RELATED CONCEPTS

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

### RELATED TASKS

"Create Master Devices"

"Play Audio Compact Discs" on page 340

"Create Audio Devices" on page 354

"Create Video Devices" on page 378

"Add Animated Buttons and Circular Sliders" on page 385

"Write an Event Handler" on page 25

"Task and Samples Cross-Reference Table" on page 445

---

## Create Master Devices

WIN

OS/2

The master volume control determines the maximum volume level of all audio output devices in the system. It sets a scale by which all subsequent volume commands are based. For instance, if the volume control sets the master volume at 50 percent, then all volume levels are cut in half.

Use the `IMMMasterAudio` class to create a master audio object. The master audio object has functions that do the following:

- Returns the current or saved setting of headphones, speakers, and master volume.
- Saves the current setting of headphones, speakers, and master volume.

- Sets the headphones and speakers on or off.
- Sets the master volume to a percent of the total volume available.

We recommend that you do not use a master volume control in your application unless it absolutely requires you to do so. This control affects all volumes in your system. One scenario that is appropriate for using a master volume control is when you are using the amplifier-mixer device class to control a complete system. In this case, you want to control all volume in the system.

### **Creating Amplifier-Mixer Devices (IMMAmpMixer)**

The visuals and functions of the amplifier-mixer device are similar to the amplifier-mixer device on your home stereo system. Components are plugged into the amplifier-mixer so that audio signals can be transferred to a pair of attached speakers, headphones, or another device. The amplifier-mixer is the center of all audio signals and provides input or output switching and sound-shaping services, such as volume, treble, or base control.

Use the IMMAmpMixer class to create an amplifier-mixer device. Its specific functions include the following:

- Controlling the balance, bass, treble, gain, and pitch of a signal
- Connecting other devices that need the above functions to the amplifier-mixer
- Turning off and on the sound that is routed through the amplifier-mixer to another device

### **Creating CD Audio Player Devices (IMMAudioCD)**

The CD audio player device's interface should look and function similarly to your home CD system as it uses the same medium, the compact disc.

A compact disc can store up to 74 minutes of 44.1-kilohertz, two-channel audio encoded as digital information. The audio compact disc, or the audio portion with both data and audio on it, is organized as tracks, where one track is typically one song. A track can be any length you want it to be, as long as it fits in the length of the disc. The length of a compact disc track is measured in minutes, seconds, and frames, where one frame is 1/75 second. It is possible to play portions of a track, starting and stopping within the accuracy of a single frame. While an application can play portions of a track, the amount of time required to seek from one track to another and locate the starting frame in question can be substantial; it can vary depending on where you are starting from. If your application calls for playing numerous sounds from a CD with precise timing, make sure they are located physically close together on the disc.

In addition to playing tracks, you can find out things about a CD, such as how many tracks it contains and how long each track is, or you can query a CD's table of contents.

The CD device class, IMMAudioCD, provides access to devices that read CDs in order to play a compact disc's digital audio data. This data format, which is digital audio, consists of sound that has been recorded as a sequence of 1's and 0's. A digital-to-analog converter recreates the original waveforms at playback.

You can perform the following functions with the IMMAudioCD class:

- Playing
- Scanning
- Tracking
- Querying a CD's table of contents

- Querying a UPC code or country code (a UPC code is a serial number that is assigned to a disc)
- Setting up to play a particular song automatically
- Forwarding or reversing to a particular track
- Forwarding or reversing to a particular location (for example, 2 minutes into track 3)

Also useful is the ability to program the order in which a CD plays its tracks. Using the IProfile class, you can record in a file the song title or track information. The IMMAudioCD object loads the profile data and plays the CD based on the contents of the data. This might be useful in a CD store, where a system allows you to listen to CDs, traversing through the various tracks in six minutes.

You can use the following command to create a CD device object:

```
#include <immcdda.hpp> // Define the header file
IMMAudioCD cdPlayer; // Define the object
cdPlayer(true); // Pass true to the device constructors so
 // the devices are opened and no additional
 // function calls are made before using
 // the device.
```

#### RELATED CONCEPTS

- “Chapter 24. Multimedia Devices” on page 335
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Play Audio Compact Discs”
- “Create Audio Devices” on page 354
- “Create Video Devices” on page 378
- “Add Animated Buttons and Circular Sliders” on page 385
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

---

## Play Audio Compact Discs

WIN

> OS/2

The interface for accessing and playing CDs should look very similar to what you are used to on your home system. It should allow you to scan, play, stop, pause, and reverse. The IMMAudioCD class works only on discs that contain CD\_DA tracks.

The application must ensure that the appropriate compact disc is in the CD drive. For example, a CD player application might simply update its track and time displays if a new disc is inserted and verified. If you try to open an IMMAudioCD object and there is not an audio CD in the CD-ROM drive, then the application sends a message that the medium is not valid. A CD-drive can only be accessed by one player at a time. An example of designing a user interface with a player panel containing a CD player device follows.

1. Define the device in the header file as follows. This excerpt is from the samp5.hpp file:



```

class CD : public IMultiCellCanvas,
 public ICommandHandler,
 public IObservable,
 public ISliderArmHandler,
 public ISelectHandler
{
public:
 CD(unsigned long windowid,
 IWindow* parent,
 IWindow* owner);
 ~CD();
protected:
 virtual bool
 command (ICommandEvent& event),
 selected(IControlEvent& event),
 moving (IControlEvent& event);
 virtual IObservable
 &dispatchNotificationEvent(const INotificationEvent&);
private:
 IMMAudioCD
 cdPlayer;
 IMMPlayerPanel
 baseButtons;
 IAnimatedButton
 trackF,
 trackB,
 scanF,
 scanB,
 eject;
 ICircularSlider
 volume;
 IStaticText
 name,
 readout;
 IRadioButton
 doorOpen,
 doorClosed;
 IMMampMixer
 *pAmpMixer;
};

```

The class CD inherits from five classes:

- The IMultiCellCanvas class allows you to arrange the push buttons, dials, and other controls on the application.
- The ICommandHandler class lets you process commands created when the user presses one of the push buttons such as **Track Reverse** or **Eject**. To process these events you override the virtual function `ICommandHandler::command`.
- The IObservable class enables the application to be notified of the behavior of the CD so that you can keep track of the current track or time elapsed. To handle notifications you override the virtual function `IObservable::dispatchNotificationEvent`.
- The ISliderArmHandler allows you to process events generated when the user turns the volume dial. To process these events you override the virtual function `ISliderArmHandler::moving`.
- The ISelectHandler class lets you handle events that occur when a user selects one of the radio buttons that opens or closes the CD drive. To handle these events you override the virtual function `ISelectHandler::selected`.

The IMMPlayerPanel class provides you with rewind, stop, pause, play, and fast forward buttons. Not only does this class create buttons and organize them, it also makes the actual function call for the button.

The IMMampMixer class lets you adjust the volume of the CD player in this example. This class is similar to a home stereo amplifier and mixer. You connect devices such as the CD player to IMMampMixer so that audio signals can be transferred to speakers or another device.

2. Create the device as follows. This excerpt is from the samp5.cpp file:

```

CD::CD(unsigned long windowid,
 IWindow* parent,
 IWindow* owner)
: IMultiCellCanvas(windowid,parent,owner),
 readout (READOUTID, this,this),
 name (CDNAMEID, this, this),
 baseButtons (BASEBUTTONID, this,this, IMMDevice::audioCD),
 trackF (TRACKFID,&baseButtons,&baseButtons,IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 trackB (TRACKBID,&baseButtons,&baseButtons,IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 scanF (SCANFID,&baseButtons,this,IRectangle(),
 ICustomButton::autoLatch |
 ICustomButton::latchable |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 scanB (SCANBID,&baseButtons,this,IRectangle(),
 ICustomButton::autoLatch |
 ICustomButton::latchable |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 eject (EJECTID,this,this,IRectangle(),
 ICustomButton::autoLatch |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 volume (VOLID, this, this, IRectangle(),
 ICircularSlider::defaultStyle() |
 ICircularSlider::proportionalTicks),
 doorOpen (OPENBTN, this, this, IRectangle(),
 IRadioButton::defaultStyle() |
 IControl::group),
 doorClosed(CLOSEDBTN, this, this),
 cdPlayer(),
 pAmpMixer(0)
{
 // Allow the CD to play without a connector.
 cdPlayer.enableConnector(IMMDevice::cdStream);
 // Allow the CD to send events
 // to handleNotificationEvent
 cdPlayer.enableNotification();
 // Add the additional button to the player panel.
 baseButtons.setPlayableDevice(&cdPlayer);
 baseButtons.addToCell(&trackB , 7, 1, 1, 1);
 baseButtons.addToCell(&trackF , 8, 1, 1, 1);
 baseButtons.addToCell(&scanB , 10, 1, 1, 1);
 baseButtons.addToCell(&scanF , 11, 1, 1, 1);
 // Put the bitmaps on the buttons.
 trackB.setBitmaps(IAnimatedButton::trackReverse);
 trackF.setBitmaps(IAnimatedButton::trackAdvance);
 scanB.setBitmaps(IAnimatedButton::scanBackward);
 scanF.setBitmaps(IAnimatedButton::scanForward);
 eject.setBitmaps(IAnimatedButton::eject);
 // Put text on the buttons. The \n cause the text to be
 // on a new line.

```

```

trackB.setText("Track\nReverse");
trackF.setText("Track\nAdvance");
scanB.setText("Scan\nBackward");
scanF.setText("Scan\nForward");
eject.setText("Eject");
doorOpen.setText("Open door");
doorClosed.setText("Close door (if possible)");
doorOpen.select();
//Set up the title
name.setText("CD Player");
//Set up the display
readout.setText("TRACK 00 MIN:SEC 00:00");
readout.setLimit(24);
volume.setArmRange (IRange(0,100));
volume.setRotationIncrement(10);
volume.setText ("Volume");
volume.setValue(100);
cdPlayer.setVolume(100);
pAmpMixer = new IMMampMixer(cdPlayer.connectedDeviceId(IMMDevice::cdStream));
pAmpMixer->enableMonitoring();
pAmpMixer->setCloseOnDestroy(false);
//Add the controls to the multicell
addToCell (&name, 2, 1, 4, 1);
addToCell (&readout, 2, 3, 4, 1);
addToCell (&doorOpen, 2, 5, 4, 1);
addToCell (&doorClosed, 2, 6, 4, 1);
addToCell (&volume, 4, 7);
addToCell (&baseButtons, 4, 9);
addToCell (&eject, 2, 9);
setColumnWidth (5, 0, true);
ISelectHandler::handleEventsFor(this);
ICommandHandler::handleEventsFor(this);
IObserver::handleNotificationsFor(cdPlayer);
ISliderArmHandler::handleEventsFor(&volume);
}

```

3. Handle the CD track and scan buttons. Call the following function in the CD constructor to start handling events generated from the CD track and scan buttons:

```
ICommandHandler::handleEventsFor(this);
```

Afterwards, override the `ICommandHandler::command` virtual function to process these events. The following code is from `samp5.cpp`:

```

bool CD::command(ICommandEvent& evt)
{
 bool
 rv = false;
 switch (evt.commandId())
 {
 case TRACKBID:
 cdPlayer.trackBackward();
 rv=true;
 break;
 case TRACKFID:
 cdPlayer.trackForward();
 rv=true;
 break;
 case SCANFID:
 if (scanF.isLatched())
 cdPlayer.startScanningForward();
 else
 cdPlayer.stop();
 rv=true;
 break;
 case SCANBID:
 if (scanB.isLatched())

```

```

 cdPlayer.startScanningBackward();
 else
 cdPlayer.stop();
 rv=true;
 break;
case EJECTID:
 if (cdPlayer.isMediaPresent())
 cdPlayer.openDoor();
 else
 {
 cdPlayer.closeDoor();
 if (cdPlayer.isMediaPresent())
 eject.unlatch();
 }
 rv = true;
 break;
}
return rv;
}
}

```

The following member functions from IMMAudioCD (unless otherwise stated) control the CD drive:

| Function                          | Description                                                                   |
|-----------------------------------|-------------------------------------------------------------------------------|
| trackBackward                     | Moves the current position backwards the passed-in number of tracks.          |
| trackForward                      | Moves the current position forwards the passed-in number of tracks.           |
| startScanningForward              | Causes the audio CD device to search forward at high speed.                   |
| startScanningBackward             | Causes the audio CD device to search backward at high speed.                  |
| stop                              | Stops playback and backward or forward scanning of the CD.                    |
| IMMRemovableMedia::isMediaPresent | Returns true if media is inserted in the device. Otherwise, it returns false. |
| IMMRemovableMedia::openDoor       | Opens the door and ejects the tray, if possible.                              |
| IMMRemovableMedia::closeDoor      | Retracts the tray and closes the door, if possible.                           |

4. Handle the notification events from the operation of the CD to update current track number and elapsed time. Call the following functions in the CD constructor to enable the IOObserver::dispatchNotificationEvent function to process notifications from cdPlayer:

```

cdPlayer.enableNotification();
IOobserver::handleNotificationsFor(cdPlayer);

```

To process these events override dispatchNotificationEvent. The following excerpt is from samp5.cpp:

```

IOobserver& CD::dispatchNotificationEvent(const INotificationEvent& event)
{
 if (event.notificationId() == IMMAudioCD::positionTimerId)
 {
 IMMTrackMinSecFrameTime* time = (IMMTrackMinSecFrameTime*)(event.eventData().asUnsignedLong
 readout.setText(IString("TRACK ") +
 IString(time->track()).rightJustify(2,'0') +
 IString(" MIN:SEC ") +

```

```

 IString(time->minutes()).rightJustify(2,'0') +
 IString(":") +
 IString(time->seconds()).rightJustify(2,'0'));
 }
 else if (event.notificationId() == IMMAudioCD::trackStartedId)
 {
 IMMTrackMinSecFrameTime* time = (IMMTrackMinSecFrameTime*)(event.eventData().asUnsignedLong());
 readout.setText(IString("TRACK ") +
 IString(time->track()).rightJustify(2,'0') +
 IString(" MIN:SEC ") +
 IString(time->minutes()).rightJustify(2,'0') +
 IString(":") +
 IString(time->seconds()).rightJustify(2,'0'));
 }
 return *this;
}

```

The notification identifiers `positionTimerId` and `trackStartedId` are sent whenever the position of the CD changes or the track of the CD changes, respectively. The `IMMTrackMinSecFrameTime` class is a data type that represents the tracks-minutes-seconds-frames time format.

5. Handle the volume slider events. Call the following function in the CD constructor to process events generated when the user turns the volume dial.

```
ISliderArmHandler::handleEventsFor(&volume);
```

To process these events override the virtual function `ISliderArmHandler::moving`. The following excerpt is from `samp5.cpp`:

```

bool CD::moving(IControlEvent& evt)
{
 bool
 result = false;
 ICircularSlider
 pSlid = (ICircularSlider)(evt.controlWindow());
 short
 val = pSlid->value();
 switch(evt.controlId())
 {
 case VOLID:
 pAmpMixer->setVolume(val);
 result = true;
 break;
 }
 return result;
}

```

The `IMMDevice::setVolume` function sets the volume of the audio channel for the device, where 0 is the least amount of volume and 100 is the most amount of volume.

6. Handle the radio buttons to open and close the CD drive. Call the following function in the CD constructor to start handling events created from the radio buttons:

```
ISelectHandler::handleEventsFor(this);
```

To process these events override the `ISelectHandler::selected` virtual function. The following excerpt is from `samp5.cpp`:

```

bool CD::selected(IControlEvent& evt)
{
 bool
 rv = false;
 switch(evt.controlId())
 {

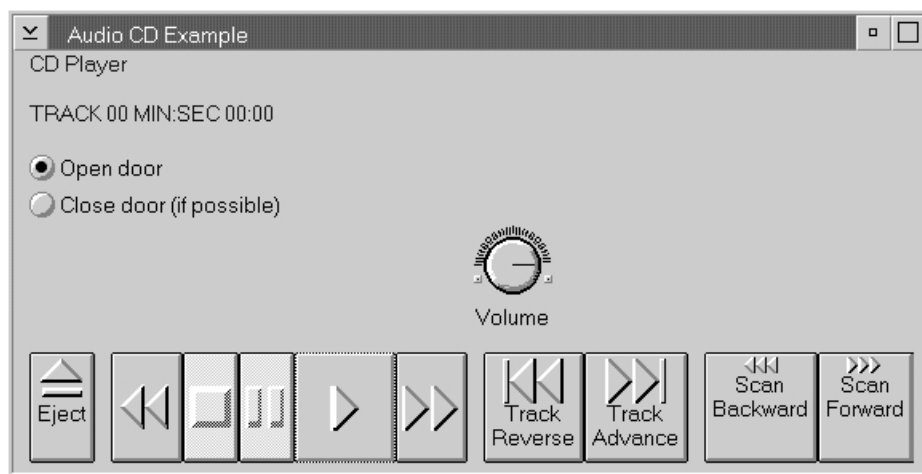
```

```

 case OPENBTN:
 // enable open cd
 cdPlayer.openDoor();
 rv = true;
 break;
 case CLOSEDBTN:
 // close cd
 cdPlayer.closeDoor();
 rv = true;
 break;
 }
 return rv;
}

```

The following figure from OS/2 shows this audio CD example. Note that the track information is displayed on the top part of the interface. The open and closed radio buttons control the CD door.



You can find the source of this example in the following files:

- samp5.cpp
- samp5.h
- samp5.hpp

### Using a MIDI Sequencer Device (IMMSequencer)

A sequencer device plays a MIDI file by sending commands to a synthesizer where the commands are converted to the sounds of a specific musical instrument. The sequencer uses the timing commands to sequence the playing of the music.

Music devices with a sequencer, such as a Casio keyboard or a drum machine (a machine that reproduces percussion sounds), can record what is being played and can play what has been recorded previously. This recording is called a *sequence*. This sequence of music notes is stored in the MIDI format.

A *sequencer* is personal computer software that allows you to record, edit, and arrange multiple tracks of MIDI data. Most sequencers let you edit the messages in a sequence and link different sequences stored in memory. This finished sequence, ready for playback, is called a *song*. If you do not want to manipulate songs already recorded with a sequencer, you can also create original songs. A sequencer lets you record any style of music you want.

The MIDI sequencer device plays a MIDI song by sending commands from a MIDI file to a synthesizer where the commands are converted to the sounds of specific instruments. The IMMSequencer class is the base class for handling a MIDI sequencer device, and it supports the MIDI standard. Thus, the sequencer controls the characteristics of the MIDI information. In addition to allowing you to load MIDI files, the IMMSequencer class inherits all of the main functions, such as play, stop, pause, and record.

You can use the following command to create a MIDI sequencer object:

```
// Define the header file
#include <immsequ.hpp>
// Define the object
IMMSequencer midiPlayer;
// Pass true to the device constructors so
// the devices are opened and no additional
// functions calls are made before using the
// device.
midiPlayer(true);
```

### Using a Waveform Audio Player Device (IMMWaveAudio)

The *waveform audio device* allows an application to play or record digital audio using files or application memory. Waveform audio devices require some form of input, that is, a file. The file contains the actual sound or waveform. The device can be opened with or without a file. If it is opened without a file, then a file is typically loaded later.

This device can use files or memory buffers. Buffering data improves performance of multimedia applications that perform numerous file input and output operations when accessing media devices. Applications that are performance-sensitive (that is, slow machines) can optimize file input and performance by buffering their data. If the data is already in the memory buffer, the operating system can transfer the record to the application's area without reading the sector from disk.

An object instantiated from IMMWaveAudio is capable of performing many tasks with a sound file. It can edit, play back, and record to name a few. In addition, the object inherits up the chain for the functions of play, stop, pause, and setFormat, plus cut, copy, and paste to, and from a memory buffer.

You can use the following command to create a waveform audio device object:

```
// Define the header file
#include <immwave.hpp>
// Define the object
IMMWaveAudio wavePlayer;
// Pass true to the device constructors so
// the devices are opened and no additional
// functions calls are made before using
// the device.
wavePlayer(true);
```

#### RELATED CONCEPTS

“Chapter 24. Multimedia Devices” on page 335

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

"Create Master Devices" on page 338  
 "Create Audio Devices" on page 354  
 "Create Video Devices" on page 378  
 "Add Animated Buttons and Circular Sliders" on page 385  
 "File: samp5.cpp"  
 "File: samp5.h" on page 352  
 "File: samp5.hpp" on page 353  
 "Write an Event Handler" on page 25  
 "Task and Samples Cross-Reference Table" on page 445

## Audio CD Example

### File: samp5.cpp

```

WIN
OS/2
This example is for Windows or OS/2.
#include "samp5.hpp"
#include "samp5.h"
#include <imgbox.hpp>
/*****
 * main
 * - Application entry point.
 *****/
int main()
{
 try
 {
 MainWindow mainWindow(WINDOWID); //Create our main window on the desktop
 IApplication::current().run(); //Get the current application and
 } // run it
 catch (IException& exc)
 {
 IMessageBox msgBox(IWindow::desktopWindow());
 msgBox.setTitle("No Multimedia");
 msgBox.show("Incompatible multimedia.",
 IMessageBox::okButton |
 IMessageBox::errorIcon);
 }
 return 0;
}
/*****
 * Class MainWindow :: MainWindow
 *****/
MainWindow::MainWindow(unsigned long windowId)
 : IFrameWindow("Audio CD Example",windowId),
 clientCanvas(CLIENTCANVASID,this,this)
{
 cd = new CD (CD_ID, &clientCanvas, this);
 setBackgroundColor(IColor(IColor::kPaleGray));
 IFont("Helv",8).setWindowFont(this);
 sizeTo(ISize(600, 300));
 setClient(cd);
 clientCanvas.setDeckOrientation(ISetCanvas::vertical);
 show();
 setFocus();
}
/*****
 * Class MainWindow :: ~MainWindow
 *****/
MainWindow::~MainWindow()
{

```



```

 if (cd)
 delete cd;
 }
 /*****
 * Class CD :: CD
 *****/
 CD::CD(unsigned long windowid,
 IWindow* parent,
 IWindow* owner)
 : IMultiCellCanvas(windowid,parent,owner),
 readout (READOUTID, this,this),
 name (CDNAMEID, this, this),
 baseButtons (BASEBUTTONID, this,this, IMMDevice::audioCD),
 trackF (TRACKFID,&baseButtons,&baseButtons,IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 trackB (TRACKBID,&baseButtons,&baseButtons,IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 scanF (SCANFID,&baseButtons,this,IRectangle(),
 ICustomButton::autoLatch |
 ICustomButton::latchable |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 scanB (SCANBID,&baseButtons,this,IRectangle(),
 ICustomButton::autoLatch |
 ICustomButton::latchable |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 eject (EJECTID,this,this,IRectangle(),
 ICustomButton::autoLatch |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 volume (VOLID, this, this, IRectangle(),
 ICircularSlider::defaultStyle() |
 ICircularSlider::proportionalTicks),
 doorOpen (OPENBTN, this, this, IRectangle(),
 IRadioButton::defaultStyle() |
 IControl::group),
 doorClosed(CLOSEDBTN, this, this),
 cdPlayer(),
 pAmpMixer(0)
 {
 // Allow the CD to play without a connector.
 cdPlayer.enableConnector(IMMDevice::cdStream);
 // Allow the CD to send events
 // to handleNotificationEvent
 cdPlayer.enableNotification();
 // Add the additional button to the player panel.
 baseButtons.setPlayableDevice(&cdPlayer);
 baseButtons.addToCell(&trackB , 7, 1, 1, 1);
 baseButtons.addToCell(&trackF , 8, 1, 1, 1);
 baseButtons.addToCell(&scanB , 10, 1, 1, 1);
 baseButtons.addToCell(&scanF , 11, 1, 1, 1);
 // Put the bitmaps on the buttons.
 trackB.setBitmaps(IAnimatedButton::trackReverse);
 trackF.setBitmaps(IAnimatedButton::trackAdvance);
 scanB.setBitmaps(IAnimatedButton::scanBackward);
 scanF.setBitmaps(IAnimatedButton::scanForward);
 eject.setBitmaps(IAnimatedButton::eject);
 // Put text on the buttons. The \n cause the text to be
 // on a new line.
 trackB.setText("Track\nReverse");
 trackF.setText("Track\nAdvance");
 scanB.setText("Scan\nBackward");
 scanF.setText("Scan\nForward");
 eject.setText("Eject");
 }

```

```

doorOpen.setText("Open door");
doorClosed.setText("Close door (if possible)");
doorOpen.select();
//Set up the title
name.setText("CD Player");
//Set up the display
readout.setText("TRACK 00 MIN:SEC 00:00");
readout.setLimit(24);
volume.setArmRange (IRange(0,100));
volume.setRotationIncrement(10);
volume.setText ("Volume");
volume.setValue(100);
cdPlayer.setVolume(100);
pAmpMixer = new IMMampMixer(cdPlayer.connectedDeviceId(IMMDevice::cdStream));
pAmpMixer->enableMonitoring();
pAmpMixer->setCloseOnDestroy(false);
//Add the controls to the multicell
addToCell (&name, 2, 1, 4, 1);
addToCell (&readout, 2, 3, 4, 1);
addToCell (&doorOpen, 2, 5, 4, 1);
addToCell (&doorClosed, 2, 6, 4, 1);
addToCell (&volume, 4, 7);
addToCell (&baseButtons, 4, 9);
addToCell (&eject, 2, 9);
setColumnWidth (5, 0, true);
ISelectHandler::handleEventsFor(this);
ICommandHandler::handleEventsFor(this);
IObserver::handleNotificationsFor(cdPlayer);
ISliderArmHandler::handleEventsFor(&volume);
}
/*****
* Class CD :: ~CD
*****/
CD::~CD()
{
 cdPlayer.stop();
 delete pAmpMixer;
}
/*****
* Class CD :: moving
*****/
bool CD::moving(IControlEvent& evt)
{
 bool
 result = false;
 ICircularSlider
 pSlid = (ICircularSlider)(evt.controlWindow());
 short
 val = pSlid->value();
 switch(evt.controlId())
 {
 case VOLID:
 pAmpMixer->setVolume(val);
 result = true;
 break;
 }
 return result;
}
/*****
* Class CD :: command
*****/
bool CD::command(ICommandEvent& evt)
{
 bool
 rv = false;
 switch (evt.commandId())
 {

```

```

case TRACKBID:
 cdPlayer.trackBackward();
 rv=true;
 break;
case TRACKFID:
 cdPlayer.trackForward();
 rv=true;
 break;
case SCANFID:
 if (scanF.isLatched())
 cdPlayer.startScanningForward();
 else
 cdPlayer.stop();
 rv=true;
 break;
case SCANBID:
 if (scanB.isLatched())
 cdPlayer.startScanningBackward();
 else
 cdPlayer.stop();
 rv=true;
 break;
case EJECTID:
 if (cdPlayer.isMediaPresent())
 cdPlayer.openDoor();
 else
 {
 cdPlayer.closeDoor();
 if (cdPlayer.isMediaPresent())
 eject.unlatch();
 }
 rv = true;
 break;
}
return rv;
}
/*****
* Class CD :: selected
*****/
bool CD::selected(IControlEvent& evt)
{
 bool
 rv = false;
 switch(evt.controlId())
 {
 case OPENBTN:
 // enable open cd
 cdPlayer.openDoor();
 rv = true;
 break;
 case CLOSEDBTN:
 // close cd
 cdPlayer.closeDoor();
 rv = true;
 break;
 }
 return rv;
}
/*****
* Class CD :: dispatchNotificationEvent
*****/
IObserver& CD::dispatchNotificationEvent(const INotificationEvent& event)
{
 if (event.notificationId() == IMMAudioCD::positionTimerId)
 {
 IMMTrackMinSecFrameTime* time = (IMMTrackMinSecFrameTime*)(event.eventData().asUnsignedLong()
 readout.setText(ISTring("TRACK ") +

```

```

 IString(time->track()).rightJustify(2,'0') +
 IString(" MIN:SEC ") +
 IString(time->minutes()).rightJustify(2,'0') +
 IString(":") +
 IString(time->seconds()).rightJustify(2,'0'));
 }
 else if (event.notificationId() == IMMAudioCD::trackStartedId)
 {
 IMMTrackMinSecFrameTime* time = (IMMTrackMinSecFrameTime*)(event.eventData().asUnsignedLong())
 readout.setText(IString("TRACK ") +
 IString(time->track()).rightJustify(2,'0') +
 IString(" MIN:SEC ") +
 IString(time->minutes()).rightJustify(2,'0') +
 IString(":") +
 IString(time->seconds()).rightJustify(2,'0'));
 }
 return *this;
}

```

#### RELATED CONCEPTS

“Chapter 24. Multimedia Devices” on page 335

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Play Audio Compact Discs” on page 340

“File: samp5.h”

“File: samp5.hpp” on page 353

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

## File: samp5.h

WIN

OS/2

This example is for Windows or OS/2.

```

#define CLIENTCANVASID 1006
#define REMOTECANVASID 1007
#define WINDOWID 1020
#define AMP_ID 1035
#define CD_ID 1036
#define ID_ICON 1037
#define ID_DUMMY 1038
#define SCANFID 4001
#define SCANBID 4002
#define TRACKFID 4003
#define TRACKBID 4004
#define EJECTID 4005
#define READOUTID 4006
#define BASEBUTTONID 4007
#define CDNAMEID 4008
#define VOLID 4009
#define BALID 4010
#define OPENBTN 4011
#define CLOSEDBTN 4012

```

#### RELATED CONCEPTS

“Chapter 24. Multimedia Devices” on page 335  
“Chapter 1. Windows” on page 1  
“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Play Audio Compact Discs” on page 340  
“File: samp5.cpp” on page 348  
“File: samp5.hpp”  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

## File: samp5.hpp

WIN

OS/2

This example is for Windows or OS/2.

```
#include "samp5.h"
#include <icolor.hpp>
#include <iframe.hpp>
#include <immccda.hpp>
#include <isetcv.hpp>
#include <ifont.hpp>
#include <imcelcv.hpp>
#include <icslider.hpp>
#include <immtime.hpp>
#include <inotifev.hpp>
#include <ianimbut.hpp>
#include <icmdhdr.hpp>
#include <istattxt.hpp>
#include <iobservr.hpp>
#include <implypn.hpp>
#include <immamix.hpp>
#include <iradiobt.hpp>
#include <iselhdr.hpp>
#include <isldahdr.hpp>
#include <ictlevt.hpp>
/*****
 * Class CD
 *****/
class CD : public IMultiCellCanvas,
 public ICommandHandler,
 public IObservable,
 public ISliderArmHandler,
 public ISelectHandler
{
public:
 CD(unsigned long windowid,
 IWindow* parent,
 IWindow* owner);
 ~CD();
protected:
 virtual bool
 command (ICommandEvent& event),
 selected(IControlEvent& event),
 moving (IControlEvent& event);
 virtual IObservable
 &dispatchNotificationEvent(const INotificationEvent&);
private:
 IMMAudioCD
```

```

 cdPlayer;
 IMMPlayerPanel
 baseButtons;
 IAnimatedButton
 trackF,
 trackB,
 scanF,
 scanB,
 eject;
 ICircularSlider
 volume;
 IStaticText
 name,
 readout;
 IRadioButton
 doorOpen,
 doorClosed;
 IMMampMixer
 *pAmpMixer;
};
/*****
* Class MainWindow
*****/
class MainWindow : public IFrameWindow
{
public:
 MainWindow(unsigned long windowId);
 ~MainWindow();
private:
 ISetCanvas
 clientCanvas;
 CD*
 cd;
};

```

#### RELATED CONCEPTS

“Chapter 24. Multimedia Devices” on page 335  
 “Chapter 1. Windows” on page 1  
 “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Play Audio Compact Discs” on page 340  
 “File: samp5.cpp” on page 348  
 “File: samp5.h” on page 352  
 “Write an Event Handler” on page 25  
 “Task and Samples Cross-Reference Table” on page 445

---

## Create Audio Devices

**WIN**

**OS/2**

The following sections discuss various ways of using the audio devices provided with the Open Class Library Multimedia Classes.

### Playing a Waveform

If you want to design an application that can record and play back an audio signal from the user’s desktop, create an IMMWaveAudio object.

To play or record audio data, a device must be able to start and stop. Also, you might want fast-forward, reverse, and pause functions. Because most people are familiar with the control panel of the typical stereo, choose that model as your application's user interface model.

### Recording a Waveform

As stated earlier, a waveform is a digital representation of a sound wave. Different formats of a waveform, such as pulse code modulation (PCM), encode sound into digital data that can be sent to an amplifier-mixer device for subsequent conversion into audio. This signal can be played through conventional speakers or earphones.

The average waveform audio driver uses PCM, 22 kiloHertz, 16 bits-per-second, and monaural as the default for 16-bit adapters. If the adapter does not support 16-bit PCM, then the resolution (bits-per-second) is downgraded to 8 bits. The types of audio resolution are 8 (multimedia), 16 (CD audio) and 24 (high-end) digital bits-per-sample. *Red Book audio* is a music industry term technically known as the CD digital audio standard for music CD audio. *Yellow Book audio* is 16-bit or 8-bit digital audio played back by the sound card. Typically, yellow book audio is stored on the personal computer as .wav files.

One of the typical uses of the waveform audio device is to digitize an input signal or sound into discrete samples for storage in a file. An example of this is recording an electronic audio mail message to tell someone about an idea, as opposed to typing a memo. An electronic audio mail application would provide the user with a simple control panel to allow the message to be recorded. Recording digitally means you get flawless sound quality that does not deteriorate.

You can record digital audio information in the format that fits your specific needs, such as for space or quality. For example, assume that a new wave audio file is created with the following code:

```
#include <immwave.hpp>
// Create the object
wavePlayer = new IMMWaveAudio(true);
// Enter begin and end time values.
wavePlayer.record(10, 20);
```

When you create the file, you might want a file that is compatible with mu-law (the compression scheme used by a telephone system). The compression scheme can change the frequency range from a telephone to CD quality.

The attributes you need to consider when recording a file are the following:

- Format and its compression algorithm (pcm, adpcm, ibmcvcsd, okiadpcm, dviadpcm, digistd, digifix, or alaw)
- Bits per sample (16-bit is considered CD quality)
- Sampling rate (for example, 22 kiloHertz)
- Number of channels (stereo or monaural)

These attributes determine the audio quality. You can even make the decision to use low-bit resolution, a low sample rate, or even monaural versus stereo on the basis of disc space and bandwidth considerations. Always set the waveform format, sampling rate, resolution, and number of channels to ensure that the waveform is created with the desired parameters.

The following is an example of code that sets these values.

```

#include <immwave.hpp>
// Create the object.
wavePlayer = new IMMWaveAudio();
wavePlayer.setBitsPerSample(Value);
// Set sampling rate
wavePlayer.setSamplesPerSecond(Value);
// Monaural is 1 (stereo is 2)
wavePlayer.setChannels(1);

```

Your application needs to define or select the recording source. The microphone is the default input device for recording waveforms.

The IMMWaveAudio class inherits the record function. An example of playing and recording a wave file follows:

1. Define the wave player device in the header file as follows. This excerpt is from samp8.hpp:

```

class WavePlayer : public IMultiCellCanvas,
 public ICommandHandler,
 public ISliderArmHandler,
 public ISelectHandler
{
public:
 WavePlayer(unsigned long windowid,
 IWindow* parent,
 IWindow* owner);

public:
 virtual bool
 command(ICommandEvent& event),
 selected(IControlEvent& event),
 moving(IControlEvent& event);
private:
 IStaticText
 infoText;
 IAnimatedButton
 playbtn,
 stopbtn,
 ffbtn,
 rewbtn,
 pausebtn,
 recordbtn;
 ICircularSlider
 volume;
 IRadioButton
 mono,
 stereo;
 IStaticText
 formatText;
 IMMWaveAudio
 wavePlayer;
};

```

The WavePlayer class inherits from four classes:

- The IMultiCellCanvas class allows you to arrange the push buttons, dials, and other controls on the application.
- The ICommandHandler class lets you process commands created from the menu or the push buttons. To process these events you override the virtual function ICommandHandler::command.
- The ISliderArmHandler allows you to process events generated when the user turns the volume dial. To process these events you override the virtual function ISliderArmHandler::moving.



- The ISelectHandler class lets you handle events that occur when a user selects one of the radio buttons that sets the number of audio channels (mono or stereo). To handle these events you override the virtual function ISelectHandler::selected.
2. Create the wave player as follows. The following excerpt is from the samp8.cpp file.

```
WavePlayer::WavePlayer(unsigned long windowid,
 IWindow* parent,
 IWindow* owner)
 :IMultiCellCanvas (windowid, parent, owner),
 volume (VOLID, this, this, IRectangle(),
 ICircularSlider::defaultStyle() |
 ICircularSlider::proportionalTicks),
 playbtn (PLAYID, this, this, IRectangle(),
 ICustomButton::latchable |
 ICustomButton::autoLatch |
 IControl::group |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 stopbtn (STOPID, this, this, IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 pausebtn (PAUSEID, this, this, IRectangle(),
 ICustomButton::latchable |
 ICustomButton::autoLatch |
 IControl::group |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 ffbtn (FFID, this, this, IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 rewbtn (REWID, this, this, IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 recordbtn (RECID, this, this, IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 infoText (INFOTXT, this, this),
 mono (MONOID, this, this, IRectangle(),
 IRadioButton::defaultStyle() |
 IControl::group),
 stereo (STEREOID, this, this),
 formatText (FORTEXTID, this, this),
 wavePlayer ()
{
 infoText.setText("Welcome to Stereo Sound - push a button");
 infoText.setForegroundColor(IColor(0, 160, 0));
 infoText.setBackgroundColor(IColor::kPaleGray);
 volume.setArmRange(IRange(0, 100));
 volume.setRotationIncrement(10);
 volume.setText("Volume");
 volume.setValue(100);
 mono.setText("Mono");
 stereo.setText("Stereo");
 stereo.select();
 formatText.setText("Select a Format:");
 addToCell(&formatText, 1, 1, 2, 1);
 addToCell(&mono, 2, 2, 1, 1);
 addToCell(&stereo, 3, 2, 1, 1);
 addToCell(&volume, 5, 4, 1, 1);
 addToCell(&rewbtn, 1, 8, 1, 1);
 addToCell(&stopbtn, 2, 8, 1, 1);
 addToCell(&pausebtn, 3, 8, 1, 1);
 addToCell(&playbtn, 4, 8, 1, 1);
 addToCell(&ffbtn, 5, 8, 1, 1);
}
```

```

addToCell(&recordbtn, 7, 8, 1, 1);
rewbtn.setBitmaps(IAnimatedButton::rewind);
ffbtn.setBitmaps(IAnimatedButton::fastForward);
pausebtn.setBitmaps(IAnimatedButton::pause);
stopbtn.setBitmaps(IAnimatedButton::stop);
playbtn.setBitmaps(IAnimatedButton::play);
recordbtn.setBitmaps(IAnimatedButton::record);
playbtn.setText ("Play");
rewbtn.setText ("Rewind");
ffbtn.setText ("Fast\nForward");
pausebtn.setText ("Pause");
stopbtn.setText ("Stop");
recordbtn.setText("Record");
playbtn.enable(false);
rewbtn.enable(false);
ffbtn.enable(false);
pausebtn.enable(false);
stopbtn.enable(false);
recordbtn.enable(false);
setBackgroundColor(IColor::kPaleGray);
setForegroundColor(IColor::kBlack);
ICommandHandler::handleEventsFor(this);
ISliderArmHandler::handleEventsFor(this);
ISelectHandler::handleEventsFor(this);
}

```

The WavePlayer constructor calls the following functions so that it can process events for the various controls in this example:

```

ICommandHandler::handleEventsFor(this);
ISliderArmHandler::handleEventsFor(this);
ISelectHandler::handleEventsFor(this);

```

### 3. Handle events as follows. This excerpt is from samp8.cpp:

```

bool WavePlayer::moving(IControlEvent& evt)
{
 bool
 result = false;
 ICircularSlider
 pSld = (ICircularSlider)(evt.controlWindow());
 short
 val = pSld->value();
 switch(evt.controlId())
 {
 case VOLID:
 wavePlayer.setVolume(val);
 result = true;
 break;
 }
 return result;
}

bool WavePlayer::command(ICommandEvent& evt)
{
 bool
 rv = false;
 switch(evt.commandId())
 {
 case MI_OPEN:
 {
 IFileDialog::Settings
 fdSettings;
 fdSettings.setTitle("Load file");
 fdSettings.setFileName("*.wav");
 IFileDialog
 fd(desktopWindow(), this, fdSettings);
 if (fd.pressedOK())
 {

```

```

 newRecord = false;
 wavePlayer.loadOnThread(fd.fileName());
 wavePlayer.setVolume(100);
 playbtn.enable();
 rewbtn.enable();
 ffbtn.enable();
 pausebtn.enable();
 stopbtn.enable();
 recordbtn.enable();
 }
 rv=true;
 break;
}
case MI_SAVE:
case MI_SAVEAS:
{
 IFileDialog::Settings
 fdSettings;
 fdSettings.setSaveAsDialog();
 fdSettings.setTitle("Save file");
 fdSettings.setFileName("*.wav");
 IFileDialog
 fd(desktopWindow(), this, fdSettings);
 if (fd.pressedOK())
 {
 wavePlayer.saveAs(fd.fileName());
 }
 rv=true;
 break;
}
case PLAYID:
{
 wavePlayer.play();
 infoText.setText("Play Mode");
 rv=true;
 break;
}
case STOPID:
{
 wavePlayer.stop();
 playbtn.enable();
 pausebtn.enable();
 ffbtn.enable();
 rewbtn.enable();
 infoText.setText("Stop Mode");
 playbtn.unlatch();
 rv=true;
 break;
}
case REWID:
{
 wavePlayer.seekToStart();
 infoText.setText("Rewind Mode");
 rv=true;
 break;
}
case FFID:
{
 wavePlayer.seekToEnd();
 infoText.setText("FF Mode");
 rv=true;
 break;
}
case PAUSEID:
{
 if (IMMDevice::paused == wavePlayer.mode())
 playbtn.latch();
}

```

```

 wavePlayer.pause();
 infoText.setText("Pause Mode");
 rv=true;
 break;
 }
 case RECID:
 {
 newRecord = false;
 recordbtn.latch();
 playbtn.enable(false);
 pausebtn.enable(false);
 ffbtn.enable(false);
 rewbtn.enable(false);
 wavePlayer.enableConnector(IMMDevice::microphones);
 wavePlayer.record();
 infoText.setText("Record Mode");
 rv=true;
 break;
 }
}
return rv;
}
bool WavePlayer::selected(IControlEvent& evt)
{
 bool
 rv = false;
 switch(evt.controlId())
 {
 case MONOID:
 if (newRecord)
 wavePlayer.setChannels(1);
 rv = true;
 break;
 case STEREOID:
 if (newRecord)
 wavePlayer.setChannels(2);
 rv = true;
 break;
 }
 return rv;
}

```

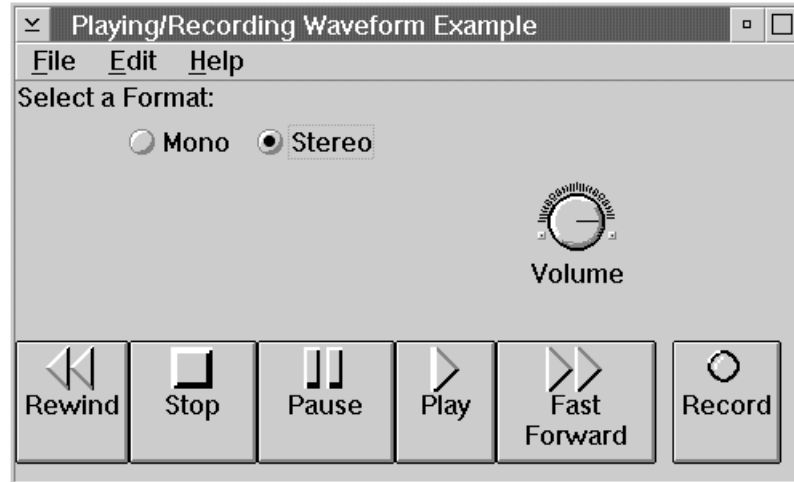
To process events generated from the volume dial override the virtual function `ISliderArmHandler::moving`. The `IMMDevice::setVolume` function sets the volume of the audio channel for the device, where 0 is the least amount of volume and 100 is the most amount of volume.

To process events generated from the menu or the push buttons override the virtual function `ICommandHandler::command`.

To process events generated from the radio buttons, override the virtual function `ISelectHandler::selected`. The `IMMConfigurableAudio::setChannels` function lets you set the number of audio channels for playing and recording:

- monaural is 1
- stereo is 2

The following figure from OS/2 demonstrates the above example:



You can find the source to the above example in the following files:

- samp8.cpp
- samp8.h
- samp8.hpp
- samp8.rc

#### **Loading the Audio or Video Device Data Files (IMMFileMedia)**

The devices that the Multimedia Classes support inherit load functions from IMMFileMedia. The following describes the ways you can load audio and video files:

- Use the load function with wait (of type CallType) as an argument. The call does not return until the system loads the file into memory. This ties up the operating system's windowing system until this call returns.
- Use the load function with nowait (of type CallType) as an argument. This creates a thread. The thread then loads the data and then notifies the attached observers when it is done. This call returns without waiting for the thread to complete. This will not tie up the operating system's windowing system.
- Use the loadOnThread function. This creates a thread. The thread then loads the data. This call does not return until the thread finishes. This does not tie up the operating system's windowing system.

#### **Using the Default Device Player (IMMPlayerPanel)**

The interface for play, pausing, and stopping should appear similar to your system at home.

You can use Open Class Library's custom player panel. The IMMPlayerPanel class creates and manages a player panel. If you create the IMMPlayerPanel without passing in a device type then you get the default buttons, which follows:

- Play
- Stop
- Pause
- Rewind
- Fast forward

If you pass in an overlay, videoDisk animation, or digital video, you get step forward and step backward buttons.

The base player panel is sufficient to control most multimedia devices.

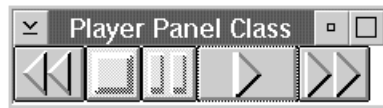
The buttons are added to an IMultiCellCanvas in the following coordinates:

| Button       | Coordinates                                       |
|--------------|---------------------------------------------------|
| Play         | (4,1) or at (5,1) if the step buttons are enabled |
| Pause        | (3,1) or at (4,1) if the step buttons are enabled |
| fastForward  | (5,1) or at (6,1) if step buttons are enabled     |
| rewind       | (1,1) or at (2,1) if step buttons are enabled     |
| stop         | (2,1) or at (3,1) if step buttons are enabled     |
| stepForward  | (7,1) if step buttons are enabled                 |
| stepBackward | (1,1) if step buttons are enabled                 |

An example of creating the custom player panel follows:

```
// Create a playable wave device
IMMWaveAudio player;
// Create the player panel and set it to control the wave audio player
IMMPlayerPanel panel(0x8008, &mainWindow, &mainWindow,
 IMMDevice::waveAudio);
panel.setPlayableDevice(player);
```

The following figure shows a custom player panel.



Notice how the stop and pause buttons are disabled when starting up the application.

### Editing a Waveform

The IMMWaveAudio class edits wave behavior. You can cut, copy, and paste to and from a memory buffer. A wave editor program allows you to record, edit, combine, and add special effects to a digital audio file. The file is not actually modified until the original file is saved. The editor allows you to mix tracks. You can use the musical editing process, for example, to correct mistakes in an artist's original interpretation or to change certain points of style before playback or final recording.

### Using Save and Save As

The IMMRecordable class provides all the common behavior for devices that support recordable media. When you save a file the binary information is stored in addition to all of the wave's attributes. For example, if you are saving a waveform, some of the attributes that are saved follow:

- Sampling rate
- Resolution
- Waveform format
- Number of channels

## Playing a Musical Instrument Digital Interface (MIDI) File

Typical user interfaces that play MIDI files have a player panel containing a MIDI device object, a menu option to load a file via a file dialog, or both. An example with a player panel follows.

1. Define the MIDI device in the header file as follows. This excerpt is from the `samp6.hpp` file:

```
class MIDI : public IMultiCellCanvas,
 public ISliderArmHandler,
 public ICommandHandler
{
public:
 MIDI(unsigned long windowid,
 IWindow* parent,
 IWindow* owner);
protected:
 virtual bool
 command(ICommandEvent& event),
 moving(IControlEvent& event);
private:
 IMMPlayerPanel
 baseButtons;
 IAnimatedButton
 loadbtn,
 rec;
 ICircularSlider
 volume;
 IStaticText
 name;
 IMMSequencer
 midiPlayer;
};
```

The `IMMSequencer` class is the base class for MIDI playback. The commands represent musical events, such as turning a note on and off, as well as timing mechanisms for specifying the duration of the note's sound.

2. Create the MIDI player as follows. The MIDI constructor is from the `samp6.cpp` file:

```
MIDI::MIDI(unsigned long windowid,
 IWindow* parent,
 IWindow* owner)
: IMultiCellCanvas (windowid, parent, owner),
 name (MIDINAMEID, this, this),
 baseButtons (BASEBUTTONID, this, this),
 volume (VOLID, this, this, IRectangle(),
 ICircularSlider::defaultStyle() |
 ICircularSlider::proportionalTicks),
 rec (RECID, &baseButtons, this, IRectangle(),
 ICustomButton::latchable |
 ICustomButton::latchable |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 loadbtn (LOADID, this, this, IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 midiPlayer ()
{
 // Add the buttons to the player panel.
 baseButtons.setPlayableDevice(&midiPlayer);
 // Put the bitmaps on the buttons.
 rec.setBitmaps(IAnimatedButton::record);
 loadbtn.setBitmaps(IAnimatedButton::eject);
 // Put text on the buttons.
 loadbtn.setText("Load");
```

```

rec.setText("Record");
// Set up the title
name.setText("MIDI Player");
name.setForegroundColor(IColor(IColor::kRed));
volume.setArmRange(IRange(0,100));
volume.setRotationIncrement(1);
volume.setText("Volume");
// Add the controls to the multicell
addToCell(&loadbtn, 1, 9, 1, 1);
addToCell(&name, 2, 1, 1, 1);
addToCell(&volume, 3, 4, 1, 3);
addToCell(&baseButtons, 2, 9, 5, 1);
// Handle events
ICommandHandler::handleEventsFor(this);
ISliderArmHandler::handleEventsFor(this);
}

```

Call the `IMMPlayableDevice::setPlayableDevice` so that the player panel `baseButtons` can control the MIDI sequencer `midPlayer`.

3. Handle events for the volume control by overriding the `ISliderArmHandler::moving` virtual function as follows. This excerpt is from `samp6.cpp`:

```

bool MIDI::moving(IControlEvent& evt)
{
 bool
 result = false;
 ICircularSlider
 pSlid = (ICircularSlider)(evt.controlWindow());
 short
 val = pSlid->value();
 switch(evt.controlId())
 {
 case VOLID:
 midiPlayer.setVolume(val);
 result = true;
 break;
 }
 return result;
}

```

4. Handle events for the menu and the animated push buttons by overriding the `ICommandHandler::command` virtual function as follows. This excerpt is from `samp6.cpp`:

```

bool MIDI::command(ICommandEvent& evt)
{
 bool
 rv = false;
 switch(evt.commandId())
 {
 case MI_OPEN:
 case LOADID:
 {
 IFileDialog::Settings
 fdSettings;
 fdSettings.setTitle("Load file");
 fdSettings.setFileName("*.mid");
 IFileDialog
 fd(desktopWindow(), this, fdSettings);
 if (fd.pressedOK())
 {
 midiPlayer.loadOnThread(fd.fileName());
 }
 rv=true;
 break;
 }
 }
}

```



```

 case PLAYID:
 {
 midiPlayer.play();
 rv=true;
 break;
 }
 }
 return rv;
}

```

The following figure displays a MIDI interface. Note that when you select the file menu option, a file dialog appears.



You can find the source to the above example in the following files:

- samp6.cpp
- samp6.h
- samp6.hpp
- samp6.rc

#### RELATED CONCEPTS

- “Chapter 24. Multimedia Devices” on page 335
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Create Master Devices” on page 338
- “Play Audio Compact Discs” on page 340
- “Create Video Devices” on page 378
- “Add Animated Buttons and Circular Sliders” on page 385
- “File: samp8.cpp”
- “File: samp8.h” on page 370
- “File: samp8.hpp” on page 371
- “File: samp8.rc” on page 372
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

## Playing/Recording Waveform Example

### File: samp8.cpp

WIN

This example is for Windows or OS/2.

```
#include "samp8.hpp"
#include "samp8.h"
#include <ifiledlg.hpp>
#include <immwave.hpp>
bool newRecord = true;
/*****
 * main
 * - Application entry point.
 *****/
int main()
{
 MainWindow mainWindow(WINDOWID);
 IApplication::current().run();
 return 0;
}
/*****
 * Class MainWindow :: MainWindow
 *****/
MainWindow::MainWindow(unsigned long windowId)
 :IFrameWindow("Playing/Recording Waveform Example", windowId),
 menuBar (windowId, this),
 myWavePlayer(WINDOWID, this, this)
{
 setClient(&myWavePlayer);
 setFocus().show();
}
/*****
 * Class WavePlayer :: WavePlayer
 *****/
WavePlayer::WavePlayer(unsigned long windowid,
 IWindow* parent,
 IWindow* owner)
 :IMultiCellCanvas (windowid, parent, owner),
 volume (VOLID, this, this, IRectangle(),
 ICircularSlider::defaultStyle() |
 ICircularSlider::proportionalTicks),
 playbtn (PLAYID, this, this, IRectangle(),
 ICustomButton::latchable |
 ICustomButton::autoLatch |
 IControl::group |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 stopbtn (STOPID, this, this, IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 pausebtn (PAUSEID, this, this, IRectangle(),
 ICustomButton::latchable |
 ICustomButton::autoLatch |
 IControl::group |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 ffbtn (FFID, this, this, IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 rewbtn (REWID, this, this, IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 recordbtn (RECID, this, this, IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 infoText (INFOTXT, this, this),
 mono (MONOID, this, this, IRectangle(),
 IRadioButton::defaultStyle() |
```

```

 IControl::group),
 stereo (STEREOID, this, this),
 formatText (FORTEXTID, this, this),
 wavePlayer ()
 {
 infoText.setText("Welcome to Stereo Sound - push a button");
 infoText.setForegroundColor(IColor(0, 160, 0));
 infoText.setBackgroundColor(IColor::kPaleGray);
 volume.setArmRange(IRange(0, 100));
 volume.setRotationIncrement(10);
 volume.setText("Volume");
 volume.setValue(100);
 mono.setText("Mono");
 stereo.setText("Stereo");
 stereo.select();
 formatText.setText("Select a Format:");
 addToCell(&formatText, 1, 1, 2, 1);
 addToCell(&mono, 2, 2, 1, 1);
 addToCell(&stereo, 3, 2, 1, 1);
 addToCell(&volume, 5, 4, 1, 1);
 addToCell(&rewbtn, 1, 8, 1, 1);
 addToCell(&stopbtn, 2, 8, 1, 1);
 addToCell(&pausebtn, 3, 8, 1, 1);
 addToCell(&playbtn, 4, 8, 1, 1);
 addToCell(&ffbtn, 5, 8, 1, 1);
 addToCell(&recordbtn, 7, 8, 1, 1);
 rewbtn.setBitmaps(IAnimatedButton::rewind);
 ffbtn.setBitmaps(IAnimatedButton::fastForward);
 pausebtn.setBitmaps(IAnimatedButton::pause);
 stopbtn.setBitmaps(IAnimatedButton::stop);
 playbtn.setBitmaps(IAnimatedButton::play);
 recordbtn.setBitmaps(IAnimatedButton::record);
 playbtn.setText ("Play");
 rewbtn.setText ("Rewind");
 ffbtn.setText ("Fast\nForward");
 pausebtn.setText ("Pause");
 stopbtn.setText ("Stop");
 recordbtn.setText("Record");
 playbtn.enable(false);
 rewbtn.enable(false);
 ffbtn.enable(false);
 pausebtn.enable(false);
 stopbtn.enable(false);
 recordbtn.enable(false);
 setBackgroundColor(IColor::kPaleGray);
 setForegroundColor(IColor::kBlack);
 ICommandHandler::handleEventsFor(this);
 ISliderArmHandler::handleEventsFor(this);
 ISelectHandler::handleEventsFor(this);
 }
}
/*****
* Class WavePlayer :: moving
*****/
bool WavePlayer::moving(IControlEvent& evt)
{
 bool
 result = false;
 ICircularSlider
 pSlid = (ICircularSlider)(evt.controlWindow());
 short
 val = pSlid->value();
 switch(evt.controlId())
 {
 case VOLID:
 wavePlayer.setVolume(val);
 result = true;
 break;
 }
}

```

```

 }
 return result;
}
/*****
* Class WavePlayer :: command
*****/
bool WavePlayer::command(ICommandEvent& evt)
{
 bool
 rv = false;
 switch(evt.commandId())
 {
 case MI_OPEN:
 {
 IFileDialog::Settings
 fdSettings;
 fdSettings.setTitle("Load file");
 fdSettings.setFileName("*.wav");
 IFileDialog
 fd(desktopWindow(), this, fdSettings);
 if (fd.pressedOK())
 {
 newRecord = false;
 wavePlayer.loadOnThread(fd.fileName());
 wavePlayer.setVolume(100);
 playbtn.enable();
 rewbtn.enable();
 ffbtn.enable();
 pausebtn.enable();
 stopbtn.enable();
 recordbtn.enable();
 }
 rv=true;
 break;
 }
 case MI_SAVE:
 case MI_SAVEAS:
 {
 IFileDialog::Settings
 fdSettings;
 fdSettings.setSaveAsDialog();
 fdSettings.setTitle("Save file");
 fdSettings.setFileName("*.wav");
 IFileDialog
 fd(desktopWindow(), this, fdSettings);
 if (fd.pressedOK())
 {
 wavePlayer.saveAs(fd.fileName());
 }
 rv=true;
 break;
 }
 case PLAYID:
 {
 wavePlayer.play();
 infoText.setText("Play Mode");
 rv=true;
 break;
 }
 case STOPID:
 {
 wavePlayer.stop();
 playbtn.enable();
 pausebtn.enable();
 ffbtn.enable();
 rewbtn.enable();
 infoText.setText("Stop Mode");
 }
 }
}

```

```

 playbtn.unlatch();
 rv=true;
 break;
 }
 case REWID:
 {
 wavePlayer.seekToStart();
 infoText.setText("Rewind Mode");
 rv=true;
 break;
 }
 case FFID:
 {
 wavePlayer.seekToEnd();
 infoText.setText("FF Mode");
 rv=true;
 break;
 }
 case PAUSEID:
 {
 if (IMMDevice::paused == wavePlayer.mode())
 playbtn.latch();
 wavePlayer.pause();
 infoText.setText("Pause Mode");
 rv=true;
 break;
 }
 case RECID:
 {
 newRecord = false;
 recordbtn.latch();
 playbtn.enable(false);
 pausebtn.enable(false);
 ffbtn.enable(false);
 rewbtn.enable(false);
 wavePlayer.enableConnector(IMMDevice::microphones);
 wavePlayer.record();
 infoText.setText("Record Mode");
 rv=true;
 break;
 }
}
return rv;
}
/*****
* Class WavePlayer :: selected
*****/
bool WavePlayer::selected(IControlEvent& evt)
{
 bool
 rv = false;
 switch(evt.controlId())
 {
 case MONOID:
 if (newRecord)
 wavePlayer.setChannels(1);
 rv = true;
 break;
 case STEREOID:
 if (newRecord)
 wavePlayer.setChannels(2);
 rv = true;
 break;
 }
 return rv;
}

```

**RELATED CONCEPTS**

- “Chapter 24. Multimedia Devices” on page 335
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

**RELATED TASKS**

- “Create Audio Devices” on page 354
- “File: samp8.h”
- “File: samp8.hpp” on page 371
- “File: samp8.rc” on page 372
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

## File: samp8.h

**WIN****OS/2**

This example is for Windows or OS/2.

```
#define CLIENTCANVASID 1006
#define PLAYID 1008
#define STOPID 1009
#define REWID 1010
#define FFID 1011
#define PAUSEID 1012
#define RECID 1013
#define VOLID 1014
#define BALID 1015
#define WINDOWID 1020
#define WAVEID 1021
#define INFOTXT 1022
#define MONOID 1023
#define STEREOID 1024
#define RIFFFORMATID 1025
#define WAVEFORMATID 1026
#define FORTEXTID 1027
#define CHANID 1028
#define PANELID 3004
#define MI_FILE 3005
#define MI_OPEN 3006
#define MI_SAVE 3007
#define MI_SAVEAS 3008
#define MI_EDIT 3009
#define MI_VOLUP 3010
#define MI_VOLDN 3011
#define MI_VOLMAX 3012
#define MI_HELP 3013
#define MI_INFO 3014
```

**RELATED CONCEPTS**

- “Chapter 24. Multimedia Devices” on page 335
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

**RELATED TASKS**

- “Create Audio Devices” on page 354
- “File: samp8.cpp” on page 365
- “File: samp8.hpp” on page 371

“File: samp8.rc” on page 372

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

## File: samp8.hpp

```
WIN
OS/2
This example is for Windows or OS/2.
#include "samp8.h"
#include <ifont.hpp>
#include <icolor.hpp>
#include <iframe.hpp>
#include <iviewport.hpp>
#include <imcelcv.hpp>
#include <icslider.hpp>
#include <isldahdr.hpp>
#include <ictlevt.hpp>
#include <istattxt.hpp>
#include <immwave.hpp>
#include <imenubar.hpp>
#include <ianimbut.hpp>
#include <isetcv.hpp>
#include <icmdhdr.hpp>
#include <iradiobt.hpp>
#include <iselhdr.hpp>
/*****
 * Class WavePlayer
 *****/
class WavePlayer : public IMultiCellCanvas,
 public ICommandHandler,
 public ISliderArmHandler,
 public ISelectHandler
{
public:
 WavePlayer(unsigned long windowid,
 IWindow* parent,
 IWindow* owner);

public:
 virtual bool
 command(ICommandEvent& event),
 selected(IControlEvent& event),
 moving(IControlEvent& event);
private:
 IStaticText
 infoText;
 IAnimatedButton
 playbtn,
 stopbtn,
 ffbtn,
 rewbtn,
 pausebtn,
 recordbtn;
 ICircularSlider
 volume;
 IRadioButton
 mono,
 stereo;
 IStaticText
 formatText;
 IMMWaveAudio
 wavePlayer;
};
/*****
```

```

* Class MainWindow
*****/
class MainWindow : public IFrameWindow
{
public:
 MainWindow(unsigned long windowId);
private:
 IMenuBar
 menuBar;
 WavePlayer
 myWavePlayer;
};

```

#### RELATED CONCEPTS

"Chapter 24. Multimedia Devices" on page 335  
 "Chapter 1. Windows" on page 1  
 "Chapter 3. Events and Event Handlers" on page 19

#### RELATED TASKS

"Create Audio Devices" on page 354  
 "File: samp8.cpp" on page 365  
 "File: samp8.h" on page 370  
 "File: samp8.rc"  
 "Write an Event Handler" on page 25  
 "Task and Samples Cross-Reference Table" on page 445

## File: samp8.rc

### WIN

This is a Windows resource script file:

```

#include "samp8.h"
WINDOWID MENUEX
BEGIN
 POPUP "&File", MI_FILE
 BEGIN
 MENUITEM "&Open...", MI_OPEN
 MENUITEM "&Save", MI_SAVE
 MENUITEM "Save&As...", MI_SAVEAS
 END
 POPUP "&Edit", MI_EDIT
 BEGIN
 MENUITEM "&Increase Volume", MI_VOLUP
 MENUITEM "&Decrease Volume", MI_VOLDN
 MENUITEM "&Maximum Volume", MI_VOLMAX
 END
 POPUP "&Help", MI_HELP
 BEGIN
 MENUITEM "&Product Information", MI_INFO
 END
END

```

### OS/2

This is an OS/2 resource script file:

```

#include "samp8.h"
MENU WINDOWID
BEGIN
 SUBMENU "~File", MI_FILE
 BEGIN
 MENUITEM "~Open...", MI_OPEN
 END

```



```

 MENUITEM "~Save", MI_SAVE
 MENUITEM "Save As...", MI_SAVEAS
 END
 SUBMENU "&Edit", MI_EDIT
 BEGIN
 MENUITEM "~Increase Volume", MI_VOLUP
 MENUITEM "~Decrease Volume", MI_VOLDN
 MENUITEM "~Maximum Volume", MI_VOLMAX
 END
 SUBMENU "~Help", MI_HELP
 BEGIN
 MENUITEM "~Product Information", MI_INFO
 END
END
END

```

#### RELATED CONCEPTS

"Chapter 24. Multimedia Devices" on page 335  
 "Chapter 1. Windows" on page 1  
 "Chapter 3. Events and Event Handlers" on page 19

#### RELATED TASKS

"Create Audio Devices" on page 354  
 "File: samp8.cpp" on page 365  
 "File: samp8.h" on page 370  
 "File: samp8.hpp" on page 371  
 "Write an Event Handler" on page 25  
 "Task and Samples Cross-Reference Table" on page 445

## Playing/Recording MIDI Example

### File: samp6.cpp

```

WIN
OS/2
This example is for Windows or OS/2.
#include "samp6.hpp"
#include "samp6.h"
/*****
* main
* - Application entry point.
*****/
int main()
{
 // Create our main window on the desktop
 MainWindow mainWindow(WINDOWID);
 // Get the current application and run it
 IApplication::current().run();
 return 0;
}
/*****
* Class MainWindow :: MainWindow
*****/
MainWindow::MainWindow(unsigned long windowId)
 : IFrameWindow ("Example MIDI Window", windowId),
 clientCanvas (CLIENTCANVASID, this, this),
 midi (MIDI_ID, &clientCanvas, this),
 menuBar (windowId, this)
{
 clientCanvas.setBackgroundColor(IColor(IColor::kPaleGray));
}

```

```

 setBackgroundColor(IColor(IColor::kPaleGray));
 IFont("Helv", 8).setWindowFont(this);
 sizeTo(ISize(500, 300));
 setClient(&midi);
 clientCanvas.setDeckOrientation(ISetCanvas::vertical);
 setFocus().show();
 }
}
/*****
* Class MainWindow :: ~MainWindow
*****/
MainWindow::~MainWindow()
{
}
/*****
* Class MIDI :: MIDI
*****/
MIDI::MIDI(unsigned long windowid,
 IWindow* parent,
 IWindow* owner)
 : IMultiCellCanvas (windowid, parent, owner),
 name (MIDINAMEID, this, this),
 baseButtons (BASEBUTTONID, this, this),
 volume (VOLID, this, this, IRectangle(),
 ICircularSlider::defaultStyle() |
 ICircularSlider::proportionalTicks),
 rec (RECID, &baseButtons, this, IRectangle(),
 ICustomButton::latchable |
 ICustomButton::latchable |
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 loadbtn (LOADID, this, this, IRectangle(),
 IWindow::visible |
 IAnimatedButton::animateWhenLatched),
 midiPlayer ()
{
 // Add the buttons to the player panel.
 baseButtons.setPlayableDevice(&midiPlayer);
 // Put the bitmaps on the buttons.
 rec.setBitmaps(IAnimatedButton::record);
 loadbtn.setBitmaps(IAnimatedButton::eject);
 // Put text on the buttons.
 loadbtn.setText("Load");
 rec.setText("Record");
 // Set up the title
 name.setText("MIDI Player");
 name.setForegroundColor(IColor(IColor::kRed));
 volume.setArmRange(IRange(0,100));
 volume.setRotationIncrement(1);
 volume.setText("Volume");
 // Add the controls to the multicell
 addToCell(&loadbtn, 1, 9, 1, 1);
 addToCell(&name, 2, 1, 1, 1);
 addToCell(&volume, 3, 4, 1, 3);
 addToCell(&baseButtons, 2, 9, 5, 1);
 // Handle events
 ICommandHandler::handleEventsFor(this);
 ISliderArmHandler::handleEventsFor(this);
}
/*****
* Class MIDI :: moving
*****/
bool MIDI::moving(IControlEvent& evt)
{
 bool
 result = false;
 ICircularSlider
 pSlid = (ICircularSlider)(evt.controlWindow());

```

```

short
 val = pSlid->value();
switch(evt.controlId())
{
 case VOLID:
 midiPlayer.setVolume(val);
 result = true;
 break;
}
return result;
}
}
/*****
* Class MIDI :: command
*****/
bool MIDI::command(ICommandEvent& evt)
{
 bool
 rv = false;
switch(evt.commandId())
{
 case MI_OPEN:
 case LOADID:
 {
 IFileDialog::Settings
 fdSettings;
 fdSettings.setTitle("Load file");
 fdSettings.setFileName("*.mid");
 IFileDialog
 fd(desktopWindow(), this, fdSettings);
 if (fd.pressedOK())
 {
 midiPlayer.loadOnThread(fd.fileName());
 }
 rv=true;
 break;
 }
 case PLAYID:
 {
 midiPlayer.play();
 rv=true;
 break;
 }
}
return rv;
}
}

```

#### RELATED CONCEPTS

“Chapter 24. Multimedia Devices” on page 335  
 “Chapter 1. Windows” on page 1  
 “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Create Audio Devices” on page 354  
 “File: samp6.h”  
 “File: samp6.hpp” on page 376  
 “File: samp6.rc” on page 378  
 “Write an Event Handler” on page 25  
 “Task and Samples Cross-Reference Table” on page 445

## File: samp6.h

WIN

OS/2

This example is for Windows or OS/2.

```
#define CLIENTCANVASID 1006
#define REMOTECANVASID 1007
#define WINDOWID 1020
#define AMP_ID 1035
#define MIDI_ID 1036
#define PLAYID 1201
#define MI_FILE 3005
#define MI_OPEN 3006
#define MI_SAVE 3007
#define MI_SAVEAS 3008
#define MI_EDIT 3009
#define MI_VOLUP 3010
#define MI_VOLDN 3011
#define MI_VOLMAX 3012
#define MI_HELP 3013
#define MI_INFO 3014
#define SCANFID 4001
#define SCANBID 4002
#define TRACKFID 4003
#define TRACKBID 4004
#define EJECTID 4005
#define READOUTID 4006
#define BASEBUTTONID 4007
#define MIDINAMEID 4008
#define VOLID 4009
#define BALID 4010
#define OPENBTN 4011
#define CLOSEDBTN 4012
#define RECID 4013
#define LOADID 4014
```

#### RELATED CONCEPTS

“Chapter 24. Multimedia Devices” on page 335  
“Chapter 1. Windows” on page 1  
“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Create Audio Devices” on page 354  
“File: samp6.cpp” on page 373  
“File: samp6.hpp”  
“File: samp6.rc” on page 378  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

## File: samp6.hpp

WIN

OS/2

This example is for Windows or OS/2.

```
#include <imenubar.hpp>
#include <icolor.hpp>
#include <iframe.hpp>
#include <immsequ.hpp>
#include <isetcv.hpp>
#include <ifont.hpp>
#include <ifiledlg.hpp>
```

```

#include <imcelcv.hpp>
#include <icslider.hpp>
#include <imtime.hpp>
#include <inotifev.hpp>
#include <ianimbut.hpp>
#include <icmdhdr.hpp>
#include <istattxt.hpp>
#include <iobservr.hpp>
#include <implypn.hpp>
#include <iradiobt.hpp>
#include <iselhdr.hpp>
#include <isldahdr.hpp>
#include <ictlevt.hpp>
/*****
 * Class MIDI
 *****/
class MIDI : public IMultiCellCanvas,
 public ISliderArmHandler,
 public ICommandHandler
{
public:
 MIDI(unsigned long windowid,
 IWindow* parent,
 IWindow* owner);
protected:
 virtual bool
 command(ICommandEvent& event),
 moving(IControlEvent& event);
private:
 IMPlayerPanel
 baseButtons;
 IAnimatedButton
 loadbtn,
 rec;
 ICircularSlider
 volume;
 IStaticText
 name;
 IMMSequencer
 midiPlayer;
};
/*****
 * Class MainWindow
 *****/
class MainWindow : public IFrameWindow,
 public ICommandHandler
{
public:
 MainWindow(unsigned long windowId);
 MainWindow();
private:
 ISetCanvas
 clientCanvas;
 MIDI
 midi;
 IMenuBar
 menuBar;
};

```

#### RELATED CONCEPTS

“Chapter 24. Multimedia Devices” on page 335

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Create Audio Devices” on page 354
- “File: samp6.cpp” on page 373
- “File: samp6.h” on page 375
- “File: samp6.rc”
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

## File: samp6.rc

#### WIN

This is a Windows resource script file:

```
#include "samp6.h"
WINDOWID MENUEX
BEGIN
 POPUP "&File", MI_FILE
 BEGIN
 MENUITEM "&Open...", MI_OPEN
 END
 END
```

#### OS/2

This is an OS/2 resource script file:

```
#include "samp6.h"
MENU WINDOWID
BEGIN
 SUBMENU "~File", MI_FILE
 BEGIN
 MENUITEM "~Open...", MI_OPEN
 END
 END
```

#### RELATED CONCEPTS

- “Chapter 24. Multimedia Devices” on page 335
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Create Audio Devices” on page 354
- “File: samp6.cpp” on page 373
- “File: samp6.h” on page 375
- “File: samp6.hpp” on page 376
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

---

## Create Video Devices

#### WIN

#### OS/2

The differences between analog and digital video are similar to the audio differences. An *analog* video is a series of squiggles that modulate an electron beam in a picture tube to paint images similar to what we see on a television.

Digitizing video freezes images into individual frames, each one a picture that can be manipulated. The frame rate is akin to the sample rate in that it explains how many times per second a video is frozen. Frame rates generally vary from 12 (animation) to 24 (film) and up. Instead of bandwidth, a video's frame rate affects how smooth the motion of objects within the video image will appear.

Video resolution defines how much information is used to describe each dot or pixel of a frame. It ranges from 8 (multimedia) to 24 (higher-end graphics).

To visualize what a digital video player device does, compare it to a VCR. Anything you can do with a VCR you can do with a digital video player device.

A digital video player device supports functions that manipulate digital video and audio files as well as digital video-only files. The audio-visual files have the extension of .avi (AVI means audio-visual interface).

The digital video device class, `IMMDigitalVideo`, includes functions that change the state of the window, query and set a device's playback speed, and change a video window's attributes. The video window is where the actual video is displayed. The window can be free-floating or in a canvas window along with the buttons to manipulate the video. This class inherits functions, such as playback, record, query, adjust speed of motion video, and modify the audio attributes of the audio stored in the video file. A sound card is required to play back the sound part of the video files.

The system-provided default video window does not provide a **Close** menu choice. If you want that capability, the digital video class provides the ability to replace the default video window with one of your choosing.

The command to create a digital video device object is as follows:

```
#include <immdigvd.hpp> // Define the header file
IMMDigitalVideo videoPlayer; // Define the object
videoPlayer(true); // Pass true to the device constructors so
 // the devices are opened and no additional
 // functions calls are made before using
 // the device.
```

### Playing Video Devices

The type of system in which you plan to play your video files is relative to the type of performance you will get when you play your files. The AVI files that store the clips can be rather large, even for a brief commercial. We recommend that you use the `loadOnThread()` function when reading in large video clips. If you use `load`, it will more than likely tie up your windowing system until the file is loaded. That can annoy your users. The `loadOnThread()` function creates a thread to do the loading, which allows a user to continue doing another task.

In addition to requiring a lot of storage, playing video files requires fast computers. A 66-megahertz, 486 system is a minimum requirement. Lesser hardware produces less realistic videos with poorer resolution. Larger objects require more processor resources to animate. This means that it takes longer to move each frame of the animation from your hard drive, or wherever it resides, to your screen.

The following summarize factors affecting playback of a video:

- Processing power of the CPU
- Throughput of data storage (for example, CD-ROM, hard disk, LAN)

- Efficiency of the display subsystem (such as the video adapter and display driver)

When a motion video device element is opened, the current position in the medium is the first playable area after any header or table of contents information.

Each frame in a motion video file has a number associated with it. The first frame is frame 0, the second frame is frame 1, and so on. The current position always indicates the frame that is about to be displayed. You can specify the play-from and play-to positions. You also give a frame position parameter to the seek command.

AVI files typically have digitized sound tracks along with their pictures. If you play the AVI file on a system with a sound card installed and turn on your speakers, you also hear the sound it contains.

The faster the machine, the faster the data processing and the playback are. Digital video is processor-intensive. Raw video requires huge amounts of memory—typically 900 kilobytes for a single frame of video, which equates to roughly 27 megabytes-per-second to record or play in real time. So, digital video is often data-compressed to make it manageable. The simplest form of video compression is a smaller frame size or slower frame rate. This is the reason most digital video used in multimedia is so small and jerky. With the slow data transfer rate of CD drives, the video must be compressed further to be able to play it back. In sum: as your frame sizes and rates go up, so do your hardware requirements.

The tracks of data on a hard drive are laid out as concentric circles, whereas a CD has a single, spiral track, like an old phonograph record. Consequently, reading data requires more processing time on a CD.

One usually designs a player panel with push buttons to play video files. Video files can either be played in the application's window or in a free-floating window. A *free-floating window* displays a video in motion where the window is separate from the main application window.

The following example code demonstrates how to play a video file in either an application window or a free-floating window:

```
// Digital Video Sample
#include <iframe.hpp>
#include <icmdhdr.hpp>
#include <iselhdr.hpp>
#include <imcelcv.hpp>
#include <immdigvd.hpp>
#include <implypn.hpp>
#include <ianimbut.hpp>
#include <iradiobt.hpp>
#include <ifiledlg.hpp>
#include <isplitcv.hpp>
#define CLIENTCANVASID 1
#define PANELID 2
#define LOADID 3
#define FREEID 4
#define STATICID 5
#define VIDEOCANVAS 6
#define WND_MAIN 7
#define MAINCANVASID 8
#define VIDEOCANVASID 9
class MainWindow : public IFrameWindow,
 public ICommandHandler,
 public ISelectHandler
```



```

{
public:
 MainWindow(unsigned long windowId);
protected:
 bool command(ICommandEvent& event),
 selected(IControlEvent& event);
private:
 ISplitCanvas mainCanvas;
 IMultiCellCanvas clientCanvas;
 ICanvas videoCanvas; // Define the canvas to place
 // controls
 IMMDigitalVideo videoPlayer; // Define the video player
 IMMPlayerPanel btnPanel; // Define the player panel
 IAnimatedButton loadBtn; // Additional button
 // to load files
 IRadioButton playFree, // Radio buttons
 // to choose free-floating
 playStatic; // or static window
};
int main()
{
 MainWindow mainWindow (WND_MAIN);
 mainWindow.setFocus();
 mainWindow.show();
 IApplication::current().run();
 return 0;
}
// MainWindow::MainWindow
MainWindow::MainWindow(unsigned long windowId)
 : IFrameWindow(windowId),
 mainCanvas(MAINCANVASID,this,this),
 clientCanvas(CLIENTCANVASID,&mainCanvas,&mainCanvas),
 videoCanvas(VIDEOCANVASID,&mainCanvas,&mainCanvas),
 btnPanel(PANELID, &clientCanvas, &clientCanvas),
 loadBtn(LOADID, &clientCanvas, &clientCanvas, IRectangle(),
 IWindow::visible | IAnimatedButton::animateWhenLatched),
 playFree(FREEID, &clientCanvas, &clientCanvas, IRectangle(),
 IRadioButton::defaultStyle() | IControl::group),
 playStatic(STATICID, &clientCanvas, &clientCanvas),
 videoPlayer(true)
{
 setTitleText("Digital Video Sample");
 mainCanvas.setOrientation(ISplitCanvas::horizontalSplit);
 videoCanvas.setBackgroundColor(IColor::black);
 btnPanel.setPlayableDevice(&videoPlayer); // Attach video
 // player to player panel
 loadBtn.setBitmaps(IAnimatedButton::eject);
 loadBtn.setText("Load");
 playFree.setText("Play video in floating window");
 playStatic.setText("Play video in static window");
 playFree.select();
 clientCanvas.addToCell(&btnPanel, 2, 7, 3, 1);
 clientCanvas.addToCell(&loadBtn, 1, 7);
 clientCanvas.addToCell(&playFree, 2, 2);
 clientCanvas.addToCell(&playStatic, 2, 4);
 ICommandHandler::handleEventsFor(&btnPanel);
 ICommandHandler::handleEventsFor(this);
 ISelectHandler::handleEventsFor(&clientCanvas);
 setClient(&mainCanvas);
 setFocus();
 show();
}
// MainWindow::command
bool MainWindow::command(ICommandEvent& evt)
{
 bool rv=false;
 switch (evt.commandId()) // Load the .avi file to play

```

```

{
 case LOADID:
 IFileDialog::Settings fdSettings;
 fdSettings.setTitle("Load file");
 fdSettings.setFileName("*.avi");
 IFileDialog fd(desktopWindow(), this, fdSettings);
 if (fd.pressedOK())
 videoPlayer.loadOnThread(fd.fileName());
 rv=true;
 break;
}
return rv;
}
// MainWindow::selected
bool MainWindow::selected(IControlEvent& evt)
{
 bool rv = false;
 switch (evt.controlId()) // Handle radio buttons to switch
 { // between playing free-floaing
 case STATICID: // or statically on the frame window
 videoPlayer.setWindow(videoCanvas);
 rv = true;
 break;
 case FREEID:
 videoPlayer.useDefaultWindow();
 rv = true;
 break;
 }
 return rv;
}

```

The following figure shows the above example playing a video file using a floating window.



The following figure shows the above example playing a video file on a canvas:



Follow these steps, as demonstrated in the above example code, to interact with video files.

1. Create buttons that will let the user interact with video files. The example code uses the `IMMPlayerPanel` class to create the rewind, stop, pause, play, and fast forward buttons. Not only does this class create buttons and organize them, it also makes the actual function call for the button. For example, the play button calls the play function on whatever device is currently set into the player panel. You can easily extend this class by using the `addToCell()` function to create new buttons:

```
class MainWindow : public IFrameWindow,
 public ICommandHandler,
 public ISelectHandler
{
 // ...
private:
 // ...
 IMMPlayerPanel btnPanel; // Define the player panel
 IAnimatedButton loadBtn; // Additional button
 // ...
};
```

This example creates an additional load button that allows the user to load a video file. This example declares the load button as an `IAnimatedButton` object. The `IAnimatedButton` class lets you animate a button by specifying a set of bitmaps to cycle through. This class provides you with predefined sets of bitmaps. The `ianimbut.hpp` header file defines these bitmaps in the enumerated type `AnimatedBitmaps`. In this example, the `loadBtn` animated button uses the bitmaps defined by `IAnimatedButton::eject`:

```
MainWindow::MainWindow(unsigned long windowId)
// ...
{
 // ...
 loadBtn.setBitmaps(IAnimatedButton::eject);
 loadBtn.setText("Load");
 // ...
}
```

2. Perform event and command handling. Because this example used the `IMMPlayerPanel` class, it did need any more code to handle video command such as play or rewind. This example needed code to implement the load button, and the radio buttons that allows the user to view a video file in a

free-floating window or a static window.

To implement the load button, this example overrides the `ICommandHandler::command()` function and creates an `IFileDialog` object:

```
bool MainWindow::command(ICommandEvent& evt)
{
 bool rv=false;
 switch (evt.commandId()) // Load the .avi file to play
 {
 case LOADID:
 IFileDialog::Settings fdSettings;
 fdSettings.setTitle("Load file");
 fdSettings.setFileName("*.avi");
 IFileDialog fd(desktopWindow(), this, fdSettings);
 if (fd.pressedOK())
 mediaPlayer.loadOnThread(fd.fileName());
 rv=true;
 break;
 }
 return rv;
}
```

To implement the radio buttons, this example overrides the `ISelectHandler::selected()` function. The example calls `setWindow()` to display the video file in the application window or calls `useDefaultWindow()` to display the video file in a free-floating window:

```
// MainWindow::selected
bool MainWindow::selected(IControlEvent& evt)
{
 bool rv = false;
 switch (evt.controlId()) // Handle radio buttons to switch
 { // between playing free-floating
 case STATICID: // or statically on the frame window
 mediaPlayer.setWindow(videoCanvas);
 rv = true;
 break;
 case FREEID:
 mediaPlayer.useDefaultWindow();
 rv = true;
 break;
 }
 return rv;
}
```

### **CD Extended-Architecture Player Device (IMMCDXA)**

The device class `IMMCDXA` provides access to devices that read CDs for the purpose of playing compact disc-extended architecture (CD-XA) data. CD-XA refers to a storage format that accommodates data that is stored in a mixture of formats. The CD-XA data is stored in part as files, in part as video, and in part as audio. The following lists the maximum amount of storage for each device:

**Video** 100 MB

**Data** 50 MB

**Audio** 20 MB

Playback control is managed by the CD-XA media device and the amplifier-mixer device.

An application for using the `IMMCDXA` class might be for video CDs or movie CDs. When giving a presentation, you might want to call up different data types at different times.

This class performs the same functions as the `IMMAudioCD` class.

#### RELATED CONCEPTS

“Chapter 24. Multimedia Devices” on page 335  
“Chapter 1. Windows” on page 1  
“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Create Master Devices” on page 338  
“Play Audio Compact Discs” on page 340  
“Create Audio Devices” on page 354  
“Add Animated Buttons and Circular Sliders”  
“Write an Event Handler” on page 25  
“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

`IMultiCellCanvas`

---

## Add Animated Buttons and Circular Sliders

WIN

OS/2

### Animated Buttons

Animated buttons are customized push buttons. For example there is a play animated button, which has a play button graphic on it. Use the `IAnimatedButton` class to create and manage the animated buttons. You can use a set of predefined bitmaps as graphics on the animated button (such as fast-forward or stop). The event handling for animated buttons is the same as with push buttons.

See the wave player example for an example of animated buttons. The rewind, stop, pause, play, fast-forward, and record buttons are all examples of animated buttons.

WIN

OS/2

### Circular Sliders

Use circular sliders for functions that a user can manipulate, such as volume and balance. You probably know how a linear slider works. A circular slider provides the same function; however, physically it looks different because it is circular. Like the dials found on your home electronics, its slider arm is shown as a radius of the dial. Outside the perimeter of the dial is a circular scale with tick marks representing incremental values the slider arm can point to. Its value can be tracked by pointing to any area on the dial and pressing the select button while moving the mouse on the desktop.

Because of its shape, the circular slider takes up less space than a linear slider and gives you more flexibility in designing a panel that has multiple controls. Use the `ICircularSlider` class to create circular sliders. Use the `ISliderArmHandler` class to handle events generated from the `ICircularSlider` objects.

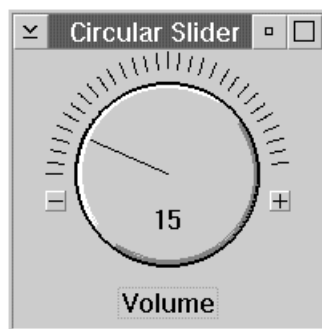
An example of a creating a circular slider follows:

```

// Circular slider example
#include <iframe.hpp>
#include <icslider.hpp>
#include <icolor.hpp>
#include <islдахdr.hpp>
#include <iapp.hpp>
class SliderArmHandler : public ISliderArmHandler
{
 public:
 SliderArmHandler();
};
/*****
* main
* - Application entry point.
*****/
int main()
{
 IFrameWindow mainWindow ("Circular Slider Example",0x1000);
 IWindow * pParent = &mainWindow;
 IWindow * pOwner = &mainWindow;
 ICircularSlider slider(0x8008,
 pParent,
 pOwner,
 IRectangle(),
 ICircularSlider::defaultStyle() |
 ICircularSlider::proportionalTicks);
 slider.setArmRange(IRange(0,100));
 slider.setRotationIncrement(1);
 slider.setText("Volume");
 SliderArmHandler *shslider = new SliderArmHandler;
 shslider->handleEventsFor(&slider);
 mainWindow.setClient(&slider);
 mainWindow.sizeTo(ISize(400,300));
 mainWindow.show();
 mainWindow.setFocus();
 IApplication::current().run();
 return 0;
}
/*****
* Class SliderArmHandler::SliderArmHandler
*****/
SliderArmHandler::SliderArmHandler()
 :ISliderArmHandler()
{}

```

The following figure shows a slider example. Note that its value is currently shown by both the detents and the number displayed on the dial.



#### RELATED CONCEPTS

*"Chapter 24. Multimedia Devices" on page 335*  
*"Chapter 1. Windows" on page 1*  
*"Chapter 3. Events and Event Handlers" on page 19*

**RELATED TASKS**

*"Create Master Devices" on page 338*  
*"Play Audio Compact Discs" on page 340*  
*"Create Audio Devices" on page 354*  
*"Create Video Devices" on page 378*  
*"Write an Event Handler" on page 25*  
*"Task and Samples Cross-Reference Table" on page 445*





---

## Chapter 25. Bidirectional Language Support

Bidirectional (BIDI) languages, such as Arabic and Hebrew, are written and read from right to left. However, while Arabic and Hebrew text have an overall right-to-left direction, they can contain left-to-right strings embedded within them, such as numbers or English text. Further, left-to-right strings can also contain right-to-left strings embedded within them, and so on. Bidirectional text can therefore contain both right-to-left and left-to-right elements. Similarly, a BIDI document should have an overall right-to-left layout, although elements within it can be left-to-right.

IBM does not provide an Arabic or Hebrew version of VisualAge for C++. You can, however, run applications developed with Open Class Library in a BIDI environment. Examples of BIDI environments include the Arabic-enabled Windows NT<sup>®</sup> operating system, the Hebrew version of the OS/2 operating system, and the AIX operating system with a language option and cultural convention of Hebrew or Arabic.

Open Class Library supports the running of two types of applications in a BIDI environment:

- Untranslated applications

These applications have little or no BIDI-specific code. Their titles, menus, prompts, and other text strings are not Arabic or Hebrew. These applications may have been written for English-speaking users and may still be targeted for English-speaking users — even when run in a BIDI environment.

You do not need to make any code changes to run a non-BIDI application in a BIDI environment. These untranslated applications will correctly maintain their normal left-to-right appearance since a non-BIDI window should not have the appearance of a BIDI document. Nevertheless, these applications can still accept BIDI input from the user, and in some cases display BIDI strings back to the user.

- Translated applications

These applications are fully enabled for Arabic and Hebrew users. They accept BIDI input, display BIDI strings, and display windows with a right-to-left layout.

You can use Open Class Library to develop BIDI-enabled applications. You can develop these applications to have a left-to-right look when you run them in a non-BIDI environment, and a right-to-left look (the look of a BIDI document) when you run them in a BIDI environment. To do so, you must add some BIDI-specific code to enable the BIDI support provided by Open Class Library.

Open Class Library externalizes its BIDI support primarily through the class `IBidiSettings` and the styles `IWindow::leftToRight` and `IWindow::rightToLeft`. `IBidiSettings` allows you to query and change the BIDI attributes for the application or an existing window. Use the BIDI styles `IWindow::rightToLeft` and `leftToRight` to set the BIDI attributes of windows as you create them.

### **Bidi Support in IWindow and Derived Classes**

Generally the operating system provides most of a standard control's right-to-left appearance and behavior, such as a radio button displaying its graphic to the right of its text, as well as its shaping of Arabic text, and handling of direction changes

between right-to-left and left-to-right elements in a BIDI string. However, some behavior is implemented by Open Class Library in the IWindow classes.

The following classes support right-to-left layout of their child windows:

- ISetCanvas
- IMultiCellCanvas
- ISplitCanvas
- IViewPort
- IToolBar
- IFrameWindow (for the positioning of its frame extensions)

Window classes that support left and right alignments will reverse their alignment when assigned a right-to-left layout, such as IStaticText and IEntryField.

OS/2

WIN

IWindow also provides a virtual `setBidiSettings` function to allow derived classes to provide control-specific code for supporting a change to their BIDI attributes. `IBidiSettings::apply` calls this virtual function. If you are providing your own class derived from IWindow, you may need to override this function and provide an implementation.

IFlyText inherits BIDI attributes from its owner window. It uses these BIDI attributes when drawing its fly-over text. Therefore, if the owner window is right-to-left, IFlyText will draw its fly-over help text as a BIDI string. IMessageBox and IMessageWindow inherit the application default BIDI attributes. They use these attributes when drawing the message text strings that they display. Therefore, if the application default BIDI attributes are right-to-left, IMessageBox and IMessageWindow will draw their message text as BIDI strings.

Menus are assigned the BIDI attributes of their owner. Therefore, if the owner window is right-to-left, its menu bar (in the case of a frame window) and pop-up menu will also have a right-to-left layout and draw their menu text as BIDI strings.

### Drawing BIDI Text

When drawing a text string in a BIDI environment, you can account for the following BIDI considerations:

- Drawing the text with a right-to-left direction as opposed to a left-to-right direction.
- Character shaping of Arabic text. (Arabic is a script language where a character can have a different shape depending on the characters preceding and following it.)
- Drawing numerals in their national form (such as Hindi) as opposed a Western form (such as Arabic).

The way you do this differs slightly by platform. These rules apply when drawing text using the 2D classes IText/IGraphicText and the obsolete 2D class IGString.

OS/2

On OS/2, you can also use `IBidiSettings` to query and set the BIDI attributes of a presentation space (or graphic context).

WIN

AIX

A string is drawn using the BIDI attributes of the window.

### Restrictions

WIN

Open Class Library provides PM-compatible versions of the following controls on Windows, to allow code written on OS/2 to be ported to Windows with minimal changes:

- IContainerControl
- INotebook
- ISlider
- IProgressIndicator
- INumericSpinButton
- ITextSpinButton

The PM-compatible versions, which are created with the pmCompatible styles of these classes, are not BIDI-enabled.

The native control versions of IContainerControl and INotebook show little difference when run in a BIDI environment.

#### RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Set and Query Default Bidirectional Attributes”

“Assign Bidirectional Attributes to Individual Windows” on page 393

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Set and Query Default Bidirectional Attributes

You can set and query the default bidi attributes for your application using the static functions `IBidiSettings::setApplicationDefaults` and `IBidiSettings::applicationDefaults`.

By setting application-level bidi attributes, you set the default bidi characteristics that Open Class Library uses to create top-level frame windows of your application — frame windows that have no parent window or use the desktop window as their parent window. All other windows inherit the bidi attributes of their parent windows. As a result, you can give all windows in your application a right-to-left look by calling `IBidiSettings::setApplicationDefaults` with the appropriate values at the start of your program.

You can do this using the following code, which tests if the application is running in a bidi environment, queries the bidi attributes of the application as assigned by the operating system, then explicitly sets the principle bidi attributes that affect the window layout and text orientation directions (it uses the default values returned for the other bidi attributes):

```

// Give all windows right-to-left behavior and appearance when
// running in an environment that supports a bidirectional
// language.
if (IBidiSettings::isBidiSupported())
{
 IBidiSettings
 applicationSettings(IBidiSettings::applicationDefaults());
 applicationSettings
 .setWindowLayout(IBidiSettings::layoutRightToLeft)
 .setTextOrientation(IBidiSettings::textRightToLeft);
 IBidiSettings::setApplicationDefaults(applicationSettings);
}

```

Window layout determines if a window has a general left-to-right or right-to-left appearance. For example, a right-to-left check box displays its graphic at the right edge of the window and its text to the left of the graphic. A right-to-left static text control reverses its alignment. A right-to-left set canvas displays its child windows from right to left. A right-to-left view port displays its vertical scroll bar to the left of the window being scrolled.

Text orientation determines if the text should have a general right-to-left or left-to-right direction. Bidi text should have a right-to-left direction.

#### > AIX

#### > OS/2

On the AIX and OS/2 operating systems, you can also set the following bidi attributes on an individual window basis:

- Text type
- Numeral display
- Symmetric swapping
- Word-by-word reordering
- Text shaping

#### WIN

You cannot set these other bidi attributes on the Windows operating system.

Calling `IBidiSettings::setApplicationDefaults` sets the default bidi attributes that Open Class Library uses on all threads of your application. As a result, if you call this function to change the default bidi attributes between calls to create individual top-level windows, you must synchronize your threads to be sure you do not accidentally create more windows with these bidi attributes than you intend.

The default window layout and text direction in a bidi environment is left-to-right if you don't call `IBidiSettings::setApplicationDefaults` or otherwise set bidi attributes through the operating system. The different platforms provide the following support:

- 

#### > OS/2

You can also set application-level bidi attributes using the `BIDIATTR` environment variable or calling the `WinSetLangInfo` API. Calling `WinSetLangInfo` to set application values overrides the value of `BIDIATTR`. Calling `IBidiSettings::setApplicationDefaults` is equivalent to calling `WinSetLangInfo` to set application values.

•

#### AIX

Motif supports bidi attributes as resources, such as `XmNtextMode` and `XmNnssMode`, that you can set in an application resource file or in the `.Xdefaults` file. For example, you can place the following line in your `.Xdefaults` file to set the layout direction resource for all widgets that support it:

```
*layoutDirection: left_to_right
```

Setting application-level bidi attributes via `IBidiSettings::setApplicationDefaults` overrides these Motif bidi resources. The value returned by `IBidiSettings::applicationDefaults` does not reflect the setting of any Motif bidi resources.

Open Class Library does not apply application-level bidi attributes to windows that it does not create. Therefore, the “wrapper” constructors (those which create an `IWindow`-derived object for an existing window, identified by either its window handle or the combination of its window identifier and parent window) do not modify the existing window. Similarly, the `IFrameWindow` constructors do not modify dialogs they load from a dialog template.

#### OS/2

OS/2 allows you to assign bidi attributes to windows and controls in a dialog template via the `BIDIPARAM` keyword.

#### WIN

Windows allows you to assign bidi attributes to windows and controls in a dialog template via the `WS_EX_RIGHT` and `WS_EX_RTLREADING` extended styles.

#### RELATED CONCEPTS

“Chapter 25. Bidirectional Language Support” on page 389

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Assign Bidirectional Attributes to Individual Windows”

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Assign Bidirectional Attributes to Individual Windows

You can also query and set the bidi attributes of individual windows. Setting the bidi attributes of a window overrides the bidi attributes the window otherwise inherits by default:

- For a top-level frame window, the attributes returned by `IBidiSettings::applicationDefaults`
- For any other window, the attributes it inherits from its parent window

The ability to set the bidi attributes of individual windows is useful for creating a bidi-enabled application that has a mixture of right-to-left and left-to-right windows. To do this, set the application-level bidi attributes to what the majority of your windows need, and set the bidi attributes of the other windows to what they need.

You can set the bidi attributes for an individual window in one of two ways:

- Create the window with either the `IWindow::leftToRight` or `IWindow::rightToLeft` style

Creating a window with the `IWindow::leftToRight` or `IWindow::rightToLeft` style sets its window orientation and text direction accordingly. The window will continue to get all other bidi attributes from either the application-level settings or its parent window.

Both styles are ignored if used in a non-bidi environment. Neither is a default style. If neither is specified, the window also gets its window orientation and text direction from the application-level bidi attributes or its parent window.

#### OS/2

#### WIN

Use `IBidiSettings` to set the bidi attributes of the window after you create it

On the Windows and OS/2 operating systems, you can change the bidi attributes of a window after you create it. You can do this using the function `IBidiSettings::apply`.

#### AIX

Motif bidi resources cannot be changed after a widget is created (for example, with `XtSetValues`). Because of the limitation, `IBidiSettings::apply` does not work on AIX. To maintain a portable application, set the application-level BIDI attributes and modify them for individual windows using the `IWindow::leftToRight` and `IWindow::rightToLeft` styles.

#### RELATED CONCEPTS

“Chapter 25. Bidirectional Language Support” on page 389

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Set and Query Default Bidirectional Attributes” on page 391

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

`IBidiSettings`

`IWindow`

---

## Chapter 26. Port and Convert Applications

---

### Convert Application Resources

The Open Class Library provides tools to help you convert Windows, OS/2, and Motif resource files.

#### Convert Bitmap and Icon Resources

WIN

OS/2

Use the **ibmpcnv** tool to convert bitmap (.bmp), icon (.ico), pointer (.ptr), or cursor( .cur) between the Windows and OS/2 operating systems.

The Motif resource compiler understands both OS/2 and Windows bitmap and icon formats.

The Open Class Library accesses bitmap and icon resources using an IResourceId (an unsigned long).

#### Convert Resource Files

The **convrc** tool on OS/2 and Windows converts resource (.rc) files between the two operating system formats. The **irccnv** tool on Motif converts a Windows resource file to the OS/2 format. Open Class Library uses the OS/2 format on the Motif operating system. There is no compiler extension to invoke **convrc** under a build. Convert your files with **convrc** either manually, in your configuration file using the run directive, or in a makefile.

To use the **convrc** on the Windows and OS/2 operating systems, use the following syntax in the VisualAge C++ command Line window:

```
convrc -option input output
```

To convert from Windows NT/95/98 to OS/2 specify the following option:

```
convrc -n input output
```

To convert from Windows 3.1 to OS/2 specify the following option:

```
convrc -w input output
```

The current platform's resource files have a .rc file extension.

The following example shows the **convrc** command to use to convert an OS/2 PM resource file, myapp.rc, to a Windows resource file:

```
convrc -o myapp.rcx myapp.rc
```

AIX

To use **irccnv** on Motif, use the following syntax:

```
irccnv -option input output
```

Since Motif's resource compiler understands the OS/2 format, no conversion is needed for OS/2 resource files.

To convert Windows to OS/2 format on Motif, use the following:

```
irccnv -o input output
```

The **convrc** and **irccnv** tools handle the following types of resources:

- accelerator table (ACCELTABLE)
- bitmap (BITMAP)
- extended menus (MENUEX)
- help tables (HELPTABLE)
- help subtable (HELPSUBTABLE)
- icon (ICON)
- menu (MENU)
- menu item (MENUITEM)
- pointers (POINTER) This is OS/2 only
- string table (STRINGTABLE)
- submenu (SUBMENU)

Also keep the following in mind:

- **convrc** and **irccnv** error checking do not flag the same errors as the resource compiler. For example, incomplete lines and missing commas produce erroneous output from **irccnv** but no error message. When you use these tools, check input files for errors using the resource compiler.
- **irccnv** processes only a single input file. For this reason, do not use wildcard (\*) characters in the input file name.
- **irccnv** does not convert included files. Convert each included file individually.
- **irccnv** does not convert dialog templates as they are not a portable solution. Use the ICanvas class instead.
- **irccnv** does not change included headers.
- The accelerator table, menu, menu item, string table, and submenu definitions cannot span multiple lines:

```
STRINGTABLE
BEGIN
 STR_HELLO,
 "Hello World"
END
```

#### RELATED CONCEPTS

“Chapter 21. Resources” on page 297

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Hello World Version 4: Adding Dialogs and Push Buttons” on page 420

Create Cross-Platform Applications

“Port Applications with Frame Windows” on page 397

“Use Native Window Controls and 3D Borders” on page 398

“Support Double-Byte Character Set and Multiple Languages” on page 399

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445



---

## Port Applications with Frame Windows

### WIN

The following limitations exist on the Windows platform with respect to child frame windows:

- A child frame window is one whose parent is not `IWindow::desktopWindow`.
- Focus will not be given to the child window and its title bar will appear as an inactive window.
- Minimizing to the parent window client area will not work. In particular, the icon may not be visible.
- Menus cannot be added to the child frame window.
- Child frame windows are not added to the task list or task bar.

We recommend that you do not use the following styles on child frame windows in portable applications:

- `IFrameWindow::maximizeButton`
- `IFrameWindow::maximized`
- `IFrameWindow::menuBar`
- `IFrameWindow::minimizeButton`
- `IFrameWindow::minimized`
- `IFrameWindow::minimizedIcon`
- `IFrameWindow::shellPosition`
- `IFrameWindow::sizingBorder`
- `IFrameWindow::systemMenu`
- `IFrameWindow::systemModal`
- `IFrameWindow::titleBar`
- `IFrameWindow::windowList`

An alternative to MDI style child windows which is more consistent with the PM, Motif, and 32-bit Windows style guides is to use secondary frame windows. You can create such windows with IBM Open Class Library by specifying the parent window as `IWindow::desktopWindow()` (or 0), and the primary frame as the owner.

Frame extensions are non-portable. Like the Windows operating system, control of the appearance of frame window decorations lies primarily with the system. Windows gives you a little more control (you can draw your own title bar, for example). Frame extensions in the non-client area are not supported in either environment. MDI is not supported on Motif and IOC does not support MDI correctly in Windows either. The main problem is attaching extensions to menus and title bar. You can not do this at all in Motif. Tool bar provides a better alternative for many of the cases where this is used.

### AIX

There is a new version of `IWindow::create()` that takes an `IWindow*` for parent and owner instead of `IWindowHandle` objects. Use this new `IWindow::create()` when you subclass an `IWindow` object. The library then ensures that the correct handle for the creation (`parent->handleForChildCreation()`) of the control is used.

A new member function returns the handle that a child of a control should use as their parent. When a control is created and an `IWindow*` is specified as the parent,

we use `handleForChildCreation()` instead of `handle()`. The default implementation of `handleForChildCreation()` just returns the handle that should be used when creating children. Any subclass that needs a different implementation, like `IViewPort`, overrides this member function and returns the appropriate handle.

#### RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Create a Frame Window” on page 4

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

`IViewPort`

`IWindowHandle`

---

## Use Native Window Controls and 3D Borders

The Open Class Library provides features to take advantage of the native look and feel of each operating system.

### WIN

#### Native Windows Controls

You can use the following classes on Windows to create either a native Windows (Win32) control or a control provided by IBM Open Class that is similar to an OS/2 control:

- `IContainerControl`
- `INotebook`
- `IProgressIndicator`
- `ISlider`
- `INumericSpinButton`
- `ITextSpinButton`

To create the OS/2-like control, construct the control object with its `pmCompatible` style. For example, to use the OS/2-like notebook instead of the Windows tab control, specify `INotebook::pmCompatible` when you construct your `INotebook` object. You need to specify this style because the native Windows control is the default in the Windows version of this product.

### OS/2

### AIX

OS/2 and AIX ignores the `pmCompatible` style.

The following lists the advantages and disadvantages of the `pmCompatible` style:

Advantages:

- Allows you to port code written for OS/2 to Windows with minimal effort, since some functions of the above control classes are not supported for native Windows controls.

Disadvantages:

- Creates controls that won't match the look of those created by other Windows applications.

#### ▶ AIX

pmCompatible controls are not supported on AIX.

#### ▶ WIN

### 3D Borders

The following controls have the border3D extended style as the default to add an etched three-dimensional border to the control:

- Combination box
- Entry field
- List box
- Multiline edit field
- Outline box
- Progress indicator
- Slider
- Spin button (ignored if you specify the noBorder or pmCompatible style)
- Static text (not the default)

#### ▶ OS/2

#### ▶ AIX

OS/2 and AIX ignores the 3D border styles.

#### RELATED CONCEPTS

“Chapter 21. Resources” on page 297

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Convert Application Resources” on page 395

“Port Applications with Frame Windows” on page 397

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Support Double-Byte Character Set and Multiple Languages

You can use one source file for your application code and then provide the double-byte character set (DBCS) and support multiple languages by using separate resource files for each of the languages you support. The Open Class Library approach includes either of the following:

- Use a single executable file with a separate .DLL for each language.
- Use separate executable files for each language (each with a separate resource file bound to it).

The following suggestions can assist you in creating DBCS-enabled applications:

- Use the canvas classes to build dialogs because you define message strings in resource files, you can translate them easily to another language without changing the source code.
- Use the IString class, which is DBCS-enabled and supports mixed strings that contain both the single-byte character set (SBCS) and DBCS characters. Objects of the IString class are essentially arrays of characters. The IString class provides functions to test the characters that make up the string. These functions help users determine whether a character is single byte or multiple byte, and whether it is a valid DBCS first byte.
- Use the IDBCSBuffer class, which ensures that the search functions do not inadvertently match the second byte of a DBCS character. The IDBCSBuffer class is derived from the IBuffer class, which holds the IString contents. The two bytes of a DBCS character will not be split.
- Use the following member functions in a DBCS-enabled application:

| Member Function | Returns True If...                                                      |
|-----------------|-------------------------------------------------------------------------|
| isCharValid     | The character at the given index is in the set of valid characters      |
| isDBCS1         | The byte at the given offset is the first byte of DBCS                  |
| isPrevDBCS      | The character preceding the one at the given offset is a DBCS character |

- Specify one of the following data type styles when you create and manage the IEntryField and IComboBox classes:

| Data Type Styles | Allows the Following Input...                                                                             |
|------------------|-----------------------------------------------------------------------------------------------------------|
| anyData          | A mixture of SBCS and DBCS characters.                                                                    |
| dbcsData         | DBCS-only data.                                                                                           |
| mixedData        | A mixture of SBCS and DBCS characters. Use this style if you plan to convert data to an EBCDIC code page. |
| sbcsData         | SBCS-only data.                                                                                           |

- Specify the appDBCSStatus style when constructing an IFrameWindow to include a DBCS status area when the frame appears in a DBCS environment. The Open Class Library automatically shares DBCS status control between a parent and child frame window.

#### RELATED CONCEPTS

“Chapter 21. Resources” on page 297

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Convert Application Resources” on page 395

“Port Applications with Frame Windows” on page 397

“Use Native Window Controls and 3D Borders” on page 398

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Chapter 27. Work with the User Interface Samples

Sample application files are provided with the IBM VisualAge C++ product. Use the Open Class Library samples to understand the classes and their use. Complete listings are included in the samples/ioc directory.

### About the Hello World Application

The Hello World sample application is divided into several versions, starting with the simplest form, version 1, and building up to the most complicated form, version 6. Each version shows you a different aspect of the Open Class Library.

Hello World Version 1: Creating a Main Window through Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics show you how to build an application called "Hello World" using the Open Class Library. However, these sample applications do not teach you C++ programming.

The samples are portable between the operating systems we provide. Application resources are targeted for the platforms on which they are to be built.

You can also take advantage of native controls by specifying a style.

### Reviewing the Conventions Used in the Samples

The Open Class Library uses conventions to enhance the usability and readability of the code. The following conventions will help you as you create applications.

- Class names begin with a capital letter. For example, all classes belonging to the Open Class Library with a global scope begin with the letter "I", as in IApplication. If a class name consists of more than one word, the first letter of each word is capitalized, such as IFrameWindow.  
In keeping with this standard, the letter "A" was chosen as the first letter (for example, AHelloWindow) for the Hello World application-defined classes. This convention helps you distinguish the Hello World application classes from the Open Class Library classes. This naming convention also helps you distinguish the classes you create from those supplied by the class library.
- Member functions begin with a lowercase letter. If a member function name consists of more than one word, the first letter of each word that follows the first word is capitalized, such as setText. In this information, single-word member functions have ClassName:: added to them; for example, the member function "show" appears as IWindow::show.

#### RELATED CONCEPTS

"Chapter 1. Windows" on page 1

"Chapter 3. Events and Event Handlers" on page 19

#### RELATED TASKS

"Hello World Version 1: Create a Main Window" on page 402

"Hello World Version 2: Add Resource Files and Frame Extensions" on page 405

"Hello World Version 3: Add Command Handlers and Menu Bars" on page 412

"Hello World Version 4: Adding Dialogs and Push Buttons" on page 420

"Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help" on page 433

"Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and

Graphics” on page 443

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Hello World Version 1: Create a Main Window

Version 1 of the Hello World sample application shows you how to create a main window and insert a text string into it using the static text control. A *static text control* is a text field, bitmap, icon, or box that you can use to label or box another control. In version 1, the “Hello World!!!” text string is inserted into a static text control.

Version 1 shows you how to do the following:

1. Create the main window
2. Create a static text control
3. Set the focus and show the main window

The main window for version 1 of the application looks like this:



### List the Hello World Version 1 Files

You can find the following files in the samples/ioc/hello1 directory:

| File         | Type of Code                                              |
|--------------|-----------------------------------------------------------|
| ahellow1.cpp | Source code for the main procedure.                       |
| ahellow1.icc | This configuration file differs for each platform.        |
| ahellow1.h   | Symbolic definitions file for the hello1 executable file. |
| ahellow1.rc  | Resource file for the hello1 executable file.             |
| makefile     | This makefile differs for each platform.                  |

### The Configuration File and Makefile

In VisualAge C++, the configuration file helps you manage your software development projects and the files associated with your project. You can find the configuration file ahellow1.icc file in the samples/ioc/hello1 directory.

In VisualAge C++ the **make** utility helps you manage your software development projects and the files associated with your project. The **make** utility uses a *makefile* to convert your source code file into an object file. The *makefile* is a special file containing a list of tasks that you provide to convert your source file. Refer to the Hello World Version 1 *makefile* in the `samples/ioc/hello1` directory.

There are two ways to invoke **make** depending on what you name your make file.

- 

**AIX**

If you have a file named *makefile* run the following command:

```
make
```

**WIN**

**OS/2**

For these platforms run the following command:

```
nmake
```

- If you have a file by a different name, for example, *makefile.ioc*, type the following:

**AIX**

```
make -f makefile.ioc
```

**WIN**

**OS/2**

```
nmake -f makefile.ioc
```

### Explore Hello World Version 1

The following sections describe each of the tasks performed by the Hello World version 1 application.

#### Create the Main Window

The first statement creates the main window, an instance of the `IFrameWindow` class, for the application. To make this class available, the application must include the `iframe.hpp` library header file, as follows:

```
#include <iframe.hpp>
```

Now that the `IFrameWindow` class is available, a variable, in this case `mainWindow`, is defined as a new object of this class. This object represents the main window of the application:

```
IFrameWindow mainWindow (WND_MAIN);
```

`WND_MAIN` is then defined in the `ahellow1.rc` file. The object represents the main window of the application.

#### Create a Static Text Control

Next, create a static text control for the "Hello, World!!!" text string. Because this control is an object of the `IStaticText` class, it includes another library header file, `istattxt.hpp`, as follows:

```
#include <istattxt.hpp>
```

Now, define another variable, `hello`, as a new object of the `IStaticText` class, which represents a static text control:

```
IStaticText hello(WND_HELLO, &mainWindow, &mainWindow);
```

`WND_HELLO` is the control ID.

The argument that follows identifies the `mainWindow` as the parent of the static text control. This positions the static text control in relation to the main window and displays it on top of the main window.

The last argument identifies the main window as the owner of the static text control. Controls notify their owner windows when events take place by using command, help, or control events. In this case, if an action is performed on the static text control, such as modifying its text string, that action is reported to the main window, which is specified as the owner. In version 1, no actions are performed on the static text control, but they are in versions 2 through 6.

### Set a Text String for the Static Text Control

After creating the static text control, give it a static text string. The `IStaticText` class is derived from the `ITextControl` class and, thus, inherits its member functions. One of those member functions, `setText`, defines the text string for the static text control. For example:

```
hello.setText(STR_HELLO);
```

Then in the `.rc` file, define the following:

```
#include "ahellow1.h"
STRINGTABLE
BEGIN
 WND_MAIN, "Hello World Sample - Version 1"
 STR_HELLO, "Hello, World!!!"
END
```

### Align the Static Text Control

Next, the `setAlignment` member function of the `IStaticText` class aligns the text string in the static text control. In this sample, it is centered both horizontally and vertically:

```
hello.setAlignment(IStaticText::centerCenter);
```

If you do not align the text string, the default placement is in the upper-left corner of the static text area.

### Set Static Text Control as the Client Window

Next, designate the static text control as the frame's client window so that the "Hello, World!!!" text string displays in the main window's client area. Use the `setClient` member function of the `IFrameWindow` class, as follows:

```
mainWindow.setClient(&hello);
```

The frame's client window is the window corresponding to the client area, which is the rectangular portion of the frame window not occupied by the other frame controls (for example, title bar, window border, and minimize and maximize buttons). Setting the static text control as the client window causes it to occupy the entire client area and to be aligned within the boundaries of that area. When the user resizes the main window, the client area (static text control in this sample) grows or shrinks.



### Set the Size of the Main Window

The following code shows you how to change the size of the main window:

```
mainWindow.setSize(ISize(400,300));
```

This sets the size of the main window to 400 pixels wide by 300 pixels high.

### Set the Focus and Show the Main Window

The following statements designate the main window as the active window and display the main window when the application runs:

```
mainWindow.setFocus();
mainWindow.show();
```

These statements use the `IFrameWindow::setFocus` and `IWindow::show` member functions:

Because `IFrameWindow` is derived from `IWindow`, the `setFocus` and `IWindow::show` member functions are inherited from the `IWindow` class. Classes inherit functions from the base classes from which they are derived. An application does not have to include those base classes. Therefore, the `IWindow` class does not need to be included in this application for its functions to be available.

### Starting Event Processing

The last statement displays the main window and starts the user interface window event processing for the application. This is accomplished by using member functions belonging to the `IApplication` and `ICurrentApplication` classes. Therefore, include another library header file, `iapp.hpp`, as follows:

```
#include <iapp.hpp>
```

The `IApplication::current` member function of the `IApplication` class returns the current application as an instance of the `ICurrentApplication` class. Next, the `ICurrentApplication::run` member function displays the main window and starts event processing for this application, using the following code:

```
IApplication::current().run();
```

#### RELATED CONCEPTS

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Create a Frame Window” on page 4

“Write an Event Handler” on page 25

“Change the Title Bar” on page 6

“Create an Information Area” on page 9

“Use Keyboard Accelerators” on page 50

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445

---

## Hello World Version 2: Add Resource Files and Frame Extensions

Version 2 of the Hello World application shows you how to use a resource file and how to add frame extensions to the application window.

A *resource file* is a file that contains data used by an application, such as text strings and icons. This data is often easier to maintain in a resource file than in the source code of an application because the resource file keeps all of the application's data together in one place.

*Frame extensions* are controls that you can add to a frame window in addition to those that are provided for you by basic frame windows. For example, in version 2, an information area is added below the client area.

Version 2 of the Hello World application extends version 1 by showing you how to:

- Get the "Hello, World!!!" text string and text for an information area
- Construct the main window and set the title and system menu icon
- Create and set the information area below the client window

The main window for version 2 of the Hello World application looks like this:



#### List the Hello World Version 2 Files

You can find the following files in the samples/ioc/hello2 directory:

| File         | Type of Code                                              |
|--------------|-----------------------------------------------------------|
| ahellow2.cpp | Source code for the main procedure and window constructor |
| ahellow2.hpp | Header file for the AHelloWindow class                    |
| ahellow2.h   | Symbolic definitions file for the hello2 executable file  |
| ahellow2.icc | This configuration file differs for each platform.        |
| ahellow2.rc  | Resource file for the hello2 executable file              |
| ahellow2.ico | Icon file for the hello2 executable file                  |
| makefile     | This makefile differs for each platform.                  |

#### The Primary Source Code File

The ahellow2.cpp file contains the source code for the main procedure and the window constructor. The tasks performed by this code are described in the following sections.

### The AHelloWindow Class Header File

The ahellow2.hpp contains the class definition and interface specifications for the AHelloWindow class, an IFrameWindow derived class, which was created specifically for this application. It is similar to an Open Class Library header file.

### The Symbolic Definitions File

The ahellow2.h file contains the symbolic definitions for this application. These definitions provide the IDs for the application main window, controls, and text strings. They are required in this version of the Hello World application, because the text strings are pulled in from a resource file.

### The Resource File

The ahellow2.rc file is the resource file for version 2 of the Hello World application. This resource file assigns an icon and three text strings to the constants defined in the ahellow2.h file. The ahellow2.h file is included in this resource file so the icon and text strings can be associated with the appropriate IDs.

VisualAge C++ provides tools for converting Windows, Motif, and OS/2-style resources. See “Convert Application Resources” on page 395 for more information about the resource file conversion tools.

### The Icon File

The ahellow2.ico file is used as both the title bar icon and the icon that displays when the application is minimized.

VisualAge C++ provides tools for converting Windows, Motif, and OS/2 bitmaps and icons. See “Convert Application Resources” on page 395 for more information about the bitmap and icon conversion tools.

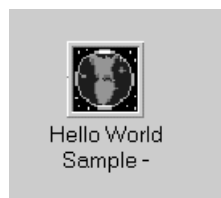
WIN

Here is how the icon appears when minimized on Windows:



OS/2

Here is how the icon appears when minimized on OS/2:



### The Advantages of the C++ File Structure

In version 1, all of the source code was intentionally put in the ahellow1.cpp file to make that version of the application simple. However, for version 2, the source code has been distributed among a variety of files to show that you can structure your applications this way.

First, the AHelloWindow class, the IFrameWindow derived class, is defined in the header file (ahellow2.hpp). Putting the class definition and interface specifications in the header file separates them from their implementation in the source code (ahellow2.cpp). This allows the class and its specifications to be used again with other applications and to be implemented in different ways. If the class definition or interface specifications change, for translation, for example, they change in only one place, the header file.

Similarly, the constant definitions file (ahellow2.h) assigns IDs to the windows and text strings in one place. Defining the constants this way allows you to use constants in a variety of places, such as the source code and the resource file, while keeping their definitions in one place. Then, if you need to change the constant definitions, you only modify the ahellow2.h file.

The advantage of placing the application's data in a resource file (ahellow2.rc) is that all of the resources are specified in one place. For example, finding and modifying text strings is easier when they are all grouped in one place, rather than searching through the source code for each one.

### **Explore Hello World Version 2**

The following sections describe each of the tasks performed by version 2 of the Hello World application. Some of the tasks are the same as those performed by version 1, but they are described again because they are performed differently in version 2.

### **Create the Main Window**

One of the major differences between version 1 and version 2 is the manner in which you create the main window. Version 1 simply creates an IFrameWindow object. However, version 2 provides its own class, AHelloWindow, to create the main window.

The AHelloWindow class is defined in the ahellow2.hpp header file and is derived from the IFrameWindow class. The IFrameWindow class is defined in the iframe.hpp library header file. Therefore, the ahellow2.hpp header file contains the following lines to make the derivation of the AHelloWindow class from the IFrameWindow class possible:

```
#ifndef _HELLOW2_
#define _HELLOW2_
#include <iframe.hpp>
#include <istattxt.hpp>
#include <iinfoa.hpp>
```

Hello World version 2 uses the compiler directive, ifndef, to prevent the ahellow2.hpp file from being included again, if it has already been included. This works because, by convention, the \_HELLOW2\_ symbol is defined in the ahellow2.hpp file as well. Both the Open Class Library and Hello World sample application use this convention in the header files.

The ahellow2.cpp file, which contains most of the source code for the application, includes the ahellow2.hpp header file to have access to the AHelloWindow class as follows:

```
#include "ahellow2.hpp"
```

The following lines in the ahellow2.cpp file create the main window by using the AHelloWindow class constructor:

```
AHelloWindow mainWindow (WND_MAIN);
```

The main window is given a value of 1000 as its window ID when the main window was created. However, instead of specifying that value in the primary source code file, we use a constant, `WND_MAIN`, which is defined in the `ahellow2.h` file, as follows:

```
#define WND_MAIN 1000
```

To have access to this definition, the primary source code file, `ahellow2.cpp`, must include the `ahellow2.h` file, as follows:

```
#include "ahellow2.h"
```

### Start Event Processing

When the main window is constructed, the following line in the `ahellow2.cpp` file gets the current application and runs it:

```
IApplication::current().run();
```

See [Start Event Processing](#) for a more detailed explanation.

### Construct the AHelloWindow Object

Version 2 constructs the main window using the `AHelloWindow` class. Here is the class constructor as it is defined in the `ahellow2.hpp` header file:

```
class AHelloWindow : public IFrameWindow
{
public:
 // Constructor
 // - construct the object with the window ID
 AHelloWindow (const unsigned long windowId);
 // Destructor
 virtual
 ~AHelloWindow() { }
private:
 IStaticText
 hello;
 IInfoArea
 infoArea;
 // Operators
 // - Assignment operator
 AHelloWindow
 &operator= (const AHelloWindow&);
};
```

In the primary source code file, `ahellow2.cpp`, version 2 uses the following lines to construct the main window:

```
AHelloWindow :: AHelloWindow(const unsigned long windowId)
: IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon,
 windowId)
,hello(WND_HELLO, this, this)
,infoArea(this)
{
 // ...
}
```

Two capabilities provided by the `IFrameWindow` class used here are not used in version 1:

- Setting the main window to the default style.

Use the `defaultStyle` member function from the `IFrameWindow` class. It returns the current default style that all applications use for frame windows. The current default style is either the original default style that is provided by the `Open`

Class Library for frame windows or a new default style that you establish by using the `setDefaultStyle` member function.

In this case, because the `setDefaultStyle` member function has not been used, the current default style is the same as the original default style, which provides a title bar, title bar icon, minimize button, maximize button, window border, window list, and an initial shell position for the window.

In this application, the title bar text and the application icon are specified in the resource file, `ahellow2`. The text string for the window title is included in the resource file, and the icon, `ahellow2.ico`, is specified.

- Displaying an icon when the main window is minimized.

The `minimizedIcon` style also inherits from the `IFrameWindow` class. This member function allows an application to use an icon to represent the application when it is minimized on the desktop. The Hello World application provides the `ahellow2.ico` icon file for this purpose.

### Create a Static Text Control

You can create a static text control to display a text string by setting `hello` equal to a new instance of the `IStaticText` class, associating an ID with the control window (`0x1010`), and making the main window the parent and owner of the control.

In our Hello World samples, however, this code is divided into separate parts and placed in different files. As shown in the following lines, `hello` is declared in the `AHelloWindow` class as an `IStaticText` object in the `ahellow2.hpp` file:

```
private:
 IStaticText
 hello;
```

In the `ahellow2.cpp` file, `hello` points to a new instance of a static text control as follows:

```
AHelloWindow :: AHelloWindow(const unsigned long windowId)
 : IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon,
 windowId)
 ,hello(WND_HELLO, this, this)
 ,infoArea(this)
{
 // ...
}
```

The `WND_HELLO` constant provides the ID for the static text control. All windows must have a unique ID, including controls. Therefore, the `ahellow2.cpp` file must include the `ahellow2.h` header file, because that is where this constant is defined:

```
#include "ahellow2.h"
```

With the `ahellow2.h` included, the ID is associated with the `WND_HELLO` constant. The following code is from the `ahellow2.h` file:

```
#define WND_HELLO 1010
```

The other two arguments (`this, this`) pass in the main window (this instance of the `AHelloWindow` class) as the parent and owner of the static text control.

See “Window Relationships” on page 2 for information about parent and owner windows.

### Set Static Text Control as the Client Window

Next, set the static text control as the client window. The following code is from the ahellow2.cpp file:

```
setClient(&hello);
```

See “Create a Static Text Control” on page 81 for an explanation of static text controls and client windows.

### Setting a Text String for the Static Text Control

After the static text control is created, the next task is to set text in it. Version 2 gets the text string from a resource file. To do this, it uses the setText member function, which it inherits from the ITextControl class. The following code is in the ahellow2.cpp file:

```
hello.setText(STR_HELLO);
```

The setText member function finds this constant string in the ahellow2.rc file and puts it into the static text control:

```
STR_HELLO, "Hello, World!!!"
```

As noted earlier, each window, even a control, must have a numeric value assigned as its ID. The resource file includes the constant definition file, so this constant definition is available. The following code is from the ahellow2.h file.

```
#define STR_HELLO 1200
```

### Create an Information Area

The following code, from the ahellow2.cpp file, creates a new instance of an information area using the IInfoArea class. This class provides a frame extension below the client window that shows information about the application.

```
AHelloWindow :: AHelloWindow(const unsigned long windowId)
 : IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon,
 windowId)
 ,hello(WND_HELLO, this, this)
 ,infoArea(this)
{
 // ...
}
```

### Set the Information Area Text

Typically, the information shown in the information area pertains to the frame menu item at which the selection cursor is currently positioned. The information is loaded from a resource file string table. A different text string displays for each menu item, changing dynamically in the information area as the cursor moves from item to item. The information area also has a special string (called “inactive text”) that displays whenever no menu item is selected.

Version 2 uses setDefaultText to set the information area’s inactive text to the same string placed in the static text control in version 1. As a result, this text appears whenever the menu is inactive. The following code is from the ahellow2.cpp file:

```
infoArea.setDefaultText(STR_INFO);
```

The setDefaultText member function finds the STR\_INFO constant in the ahellow2.rc file and puts it into the information area as follows:

```
STR_INFO, "Use Alt-F4 to Close Window"
```

The STR\_INFO constant is associated with a string ID, hexadecimal value 0x1220, in the ahellow2.h constant definition file. The resource file includes the constant definition file, so this constant definition is available as follows:

```
#define STR_INFO 1220
```

### Align the Static Text Control

As in version 1, the static text control for the client area is centered both horizontally and vertically in the static text control. The following code is from the ahellow2.cpp file:

```
hello.setAlignment(IStaticText::centerCenter);
```

#### RELATED CONCEPTS

- “Chapter 21. Resources” on page 297
- “Window Relationships” on page 2
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Convert Application Resources” on page 395
- Start Event Processing
- Create a Static Text Control
- “Hello World Version 1: Create a Main Window” on page 402
- “Write an Event Handler” on page 25
- “Create a List Box” on page 119
- “Create a Push Button” on page 99
- “Create a Static Text Control” on page 81
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

---

## Hello World Version 3: Add Command Handlers and Menu Bars

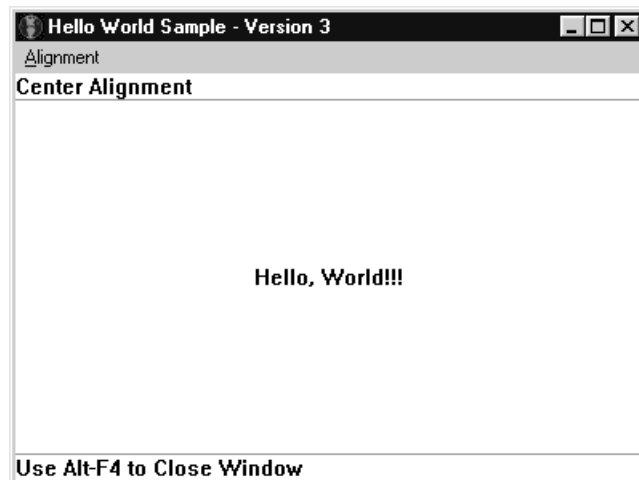
Version 3 provides a menu bar with an **Alignment** choice. A *menu bar* is the area near the top of a window, below the title bar and above the client area of the window, which contains a list of choices. By selecting the **Alignment** choice, the user can display a pull-down menu and align the “Hello, World!!!” text string to the left, right, or center. In addition, this version adds a status area to show the status of the text string and an event handler for the menu bar and the pull-down menu.

In covering these topics, this section shows you how to:

- Create a status line to show the status of the text string alignment
- Use an event handler
- Add a menu bar
- Set an initial check mark in the pull-down menu
- Add command processing (event handling) to align a text string

The main window for version 3 of the Hello World application looks like this:





### Listing the Hello World Version 3 Files

You can find the following files in the samples/ioc/hello3 directory:

| File        | Type of Code                                                                        |
|-------------|-------------------------------------------------------------------------------------|
| ahello3.cpp | Source code for the main procedure, main window constructor, and command processing |
| ahello3.hpp | Header file for the AHelloWindow and ACommandHandler classes                        |
| ahello3.h   | Symbolic definitions file for the hello3.exe file                                   |
| ahello3.icc | This configuration file differs on each platform.                                   |
| ahello3.rc  | Resource file for the hello3.exe file                                               |
| ahello3.ico | Icon file for the hello3.exe file                                                   |
| makefile    | This makefile differs on each platform.                                             |

#### The Primary Source Code File

The ahello3.cpp file contains the source code for the main procedure and the AHelloWindow and ACommandHandler classes. The tasks performed by this code are described in the following sections.

#### The AHelloWindow Class Header File

The ahello3.hpp file, like the ahello2.hpp file, contains the class definitions and interface specifications for the AHelloWindow and ACommandHandler classes, with a few modifications for version 3.

#### The Symbolic Definitions File

The ahello3.h file contains the definitions for this application. These definitions provide the IDs for the application window components.

For version 3, the symbolic definition file contains a new window ID (WND\_STATUS) for the status area and three new string IDs (STR\_CENTER, STR\_LEFT, and STR\_RIGHT) for the text strings used in the status area. In addition, menu IDs (MI\_ALIGNMENT, MI\_CENTER, MI\_LEFT, and MI\_RIGHT) have been added for the menu bar **Alignment** choice and the **Center**, **Left**, and **Right** choices in the pull-down menu.

### The Resource File

Version 3 provides a resource file, `ahellow3.rc`. This resource file assigns an icon and several text strings with the constants defined in the `ahellow3.h` file. It also contains the text strings for the menu bar. The `ahellow3.h` file is included in this resource file so the icon and text strings can be associated with the appropriate IDs.

The resource file for version 3 contains two primary additions. The first is the text strings that are assigned to the new string constants that were defined in `ahellow3.h`. These text strings are used in the status area to show the state of the static "Hello, World!!!" text string in the client area. For example, when the main window is first displayed, the "Center Alignment" text string is shown in the status area.

The second addition provides the text that appears on the menu bar (**Alignment**) and pull-down menu (**Left**, **Center**, and **Right**), indicating which choices are available. Each text string is assigned to a constant, also defined in `ahellow3.h`.

In an AIX or OS/2 resource file, the tilde (~) to the left of a letter in each text string indicates that the user can select those letters to select a menu item. This is an ampersand (&) in the Windows `.rc` file. In addition, the Windows `.rc` file uses the extended menu as follows:

```
WND_MAIN MENUEX
BEGIN
 POPUP "&Alignment" , MI_ALIGNMENT
 BEGIN
 MENUITEM "&Left" , MI_LEFT
 MENUITEM "&Center" , MI_CENTER
 MENUITEM "&Right" , MI_RIGHT
 END
END
```

These are shortcut keys for the application. For example, the sample bevaves in the same way if you either press **R** or choose **Right** from the pull-down menu.

VisualAge C++ provides tools for converting Windows, AIX, and OS/2-style resources. See "Convert Application Resources" on page 395 for more information about the resource file conversion tools.

### The Icon File

The `ahellow3.ico` file is used as both the title bar icon and the icon that displays when the application is minimized. This icon is the same as for version 2.

VisualAge C++ provides tools for converting Windows, AIX, and OS/2 bitmaps and icons. See "Convert Application Resources" on page 395 for more information about the bitmap and icon conversion tools.

### Explore Hello World Version 3

The following sections describe each of the tasks performed by version 3 of the Hello World application that have not been described for previous versions.

#### Construct the `AHelloWindow` Object

Version 3 has made the following additions to the main window:

- Create a status line
- Create a menu bar
- Set an initial check mark in the pull-down menu

- Align a text string
- Set `ACommandHandler` the event handler
- Destruct the main window

The following sections describe these additions.

### Create a Status Line

The status line shows the text string alignment status. Use the `IStaticText` class to create the static text control to display a text string in a status area. The *status area* is a small rectangular area that is usually located at the top of a window, below the menu bar.

In the `ahellow3.cpp` file, an `IStaticText` object, called `statusLine`, is created with this instance of the `AHelloWindow` class as the parent and owner as follows:

```
AHelloWindow :: AHelloWindow(const unsigned long windowId)
 :IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon,
 windowId)
 ,menuBar(windowId, this)
 ,statusLine(WND_STATUS,this,this)
 ,hello(WND_HELLO, this, this)
 ,infoArea(this)
 ,commandHandler(this)
{
 // ...
}
```

The `WND_STATUS` constant provides the window ID for this static text control. This constant is defined in `ahellow3.h`.

### Specify the Location and Height of the Status Area

Use the `IFrameWindow` member function `addExtension` in the `ahellow3.cpp` file to specify where the status area is positioned and how high it is. For example:

```
AHelloWindow :: AHelloWindow(const unsigned long windowId)
 :IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon,
 windowId)
 ,menuBar(windowId, this)
 ,statusLine(WND_STATUS,this,this)
 ,hello(WND_HELLO, this, this)
 ,infoArea(this)
 ,commandHandler(this)
{
 // ...
 addExtension(&statusLine,
 IFrameWindow::aboveClient,
 IFont(&statusLine).maxCharHeight());
 // ...
}
```

The `aboveClient` argument of the `Location` enumeration specifies that the static text control displays the status area above the client window.

The `maxCharHeight` member function returns the status area's maximum height, based on the current font.

### Create a Menu Bar

Now you can create the `Alignment` menu bar to display the **Left**, **Center**, and **Right** choices. In the header file, `ahellow3.hpp`, `menuBar` is defined as an instance of the `IMenuBar` class as follows:

```

class AHelloWindow : public IFrameWindow
{
 // ...
private:
 IMenuBar
 menuBar;
 // ...
};

```

The ahellow3.cpp file uses menuBar to create a new instance of that class in the main window, as follows:

```

AHelloWindow :: AHelloWindow(const unsigned long windowId)
 :IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon,
 windowId)
 ,menuBar(windowId, this)
 ,statusLine(WND_STATUS,this,this)
 ,hello(WND_HELLO, this, this)
 ,infoArea(this)
 ,commandHandler(this)
{
 // ...
}

```

The WND\_MAIN argument identifies the following menu in the ahellow3.rc resource file:

```

MENU WND_MAIN
BEGIN
 SUBMENU "~Alignment", MI_ALIGNMENT
 BEGIN
 MENUITEM "~Left", MI_LEFT
 MENUITEM "~Center", MI_CENTER
 MENUITEM "~Right", MI_RIGHT
 END
END

```

The window ID for the menu must match the window ID of the frame window.

This menu puts one choice, **Alignment**, on the menu bar, and provides a pull-down menu with three choices: **Left**, **Center**, and **Right**.

In addition, the MI\_ALIGNMENT, MI\_LEFT, MI\_CENTER, and MI\_RIGHT menu item attributes correspond to those in the resource file's string table:

```

MI_ALIGNMENT,"Alignment Menu"
MI_CENTER, "Set Center Alignment"
MI_LEFT, "Set Left Alignment"
MI_RIGHT, "Set Right Alignment"

```

When the user moves the selection cursor over each menu item, the text string associated with that menu item displays in the information area below the client window. For example, when the cursor is on the **Right** menu item, the text string "Set Right Alignment" appears in the information area. For this to work, the string ID must match the corresponding menu item ID.

#### Set an Initial Check Mark in the Pull-Down Menu

The pull-down menu that displays when **Alignment** is selected on the menu bar contains three choices for aligning the "Hello, World!!!" text string: **Left**, **Center**, and **Right**. Because this text string is aligned in the center of the client area when the application is created, there should be a check mark next to **Center** the first time the pull-down menu displays.

The `checkItem` member function of the `IMenuBar` class lets you place a check mark on a pull-down menu choice. The following line, from `ahellow3.cpp`, places a check mark on **Center**:

```
AHelloWindow &
AHelloWindow :: setTextAlignment(const Alignment alignment)
{
 switch(alignment)
 {
 // ...
 case center:
 hello.setAlignment(IStaticText::centerCenter);
 statusLine.setText(STR_CENTER);
 menuBar.checkItem(MI_CENTER);
 menuBar.uncheckItem(MI_LEFT);
 menuBar.uncheckItem(MI_RIGHT);
 break;
 // ...
 }
 return (*this);
}
```

The `MI_CENTER` constant is defined in the `ahellow3.rc` resource file as the “Center” text string for the menu. Do not confuse this with the `MI_CENTER` menu item attribute defined in the string table, which is used only by the information area.

### Stop Handling Command Events

After your application runs, you need to stop handling command events for the frame window and delete the objects you created using the new operator. The `AHelloWindow` destructor does this in the `ahellow3.cpp` file do this:

```
AHelloWindow :: ~AHelloWindow()
{
 commandHandler.stopHandlingEventsFor(this);
}
```

### Align a Text String

This section shows you how to associate commands with the menu items to align the text string.

This sample shows command processing for one of the menu items. This code, from `ahellow3.cpp`, calls the `AHelloWindow::setTextAlignment` function to center-align the “Hello, World!!!” text string in the client window:

```
bool ACommandHandler :: command(ICommandEvent & cmdEvent)
{
 bool eventProcessed(true);
 switch (cmdEvent.commandId())
 {
 case MI_CENTER:
 frame->setTextAlignment(AHelloWindow::center);
 break;
 case MI_LEFT:
 frame->setTextAlignment(AHelloWindow::left);
 break;
 case MI_RIGHT:
 frame->setTextAlignment(AHelloWindow::right);
 break;
 default:
 eventProcessed=false;
 }
 return(eventProcessed);
}
```

The following code shows the `setTextAlignment` function from the `AHelloWindow` class:

```
AHelloWindow &
AHelloWindow :: setTextAlignment(const Alignment alignment)
{
 switch(alignment)
 {
 case left:
 hello.setAlignment(IStaticText::centerLeft);
 statusLine.setText(STR_LEFT);
 menuBar.uncheckItem(MI_CENTER);
 menuBar.checkItem(MI_LEFT);
 menuBar.uncheckItem(MI_RIGHT);
 break;
 case center:
 hello.setAlignment(IStaticText::centerCenter);
 statusLine.setText(STR_CENTER);
 menuBar.checkItem(MI_CENTER);
 menuBar.uncheckItem(MI_LEFT);
 menuBar.uncheckItem(MI_RIGHT);
 break;
 case right:
 hello.setAlignment(IStaticText::centerRight);
 statusLine.setText(STR_RIGHT);
 menuBar.uncheckItem(MI_CENTER);
 menuBar.uncheckItem(MI_LEFT);
 menuBar.checkItem(MI_RIGHT);
 break;
 }
 return (*this);
}
```

The preceding code does the following:

- Uses the `IStaticText` `setAlignment` member function to center the static text control vertically and align it on the left horizontally
- Sets the appropriate text string in the status area (left alignment)
- Uses the `uncheckItem` member function to remove any existing check marks from the **Center** and **Right** menu items
- Uses the `checkItem` member function to set a check mark on the **Left**
- Returns true and ends

#### Add Text for a Status Line

The status area text strings are specified in the resource file, as shown in the following code:

```
MI_CENTER, "Set Center Alignment"
MI_LEFT, "Set Left Alignment"
MI_RIGHT, "Set Right Alignment"
```

The following code, from `ahellow3.cpp`, gets the "Center Alignment" text string from the resource file and puts it in the static text control for the status area:

```
AHelloWindow &
AHelloWindow :: setTextAlignment(const Alignment alignment)
{
 switch(alignment)
 {
 // ...
 case center:
 hello.setAlignment(IStaticText::centerCenter);
 statusLine.setText(STR_CENTER);
 menuBar.checkItem(MI_CENTER);
 menuBar.uncheckItem(MI_LEFT);
 menuBar.uncheckItem(MI_RIGHT);
 }
}
```

```

 break;
 // ...
}
return (*this);
}

```

### Instantiate the Command Handler

In version 3, the `AHelloWindow` class contains an `ICommandHandler` derived class, called `ACommandHandler`. This is necessary because, for the first time, this application handles events. In this case, the events are the commands that align the “Hello, World!!!” text string.

The following code in `ahellow3.cpp` creates a command handler from the `ACommandHandler` class:

```

AHelloWindow :: AHelloWindow(const unsigned long windowId)
 :IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon,
 windowId)
 ,menuBar(windowId, this)
 ,statusLine(WND_STATUS,this,this)
 ,hello(WND_HELLO, this, this)
 ,infoArea(this)
 ,commandHandler(this)
{
 // ...
 commandHandler.handleEventsFor(this);
 // ...
}

```

The second line of code in the preceding example contains the `handleEventsFor` member function inherited from the `ICommandHandler` class. Use this member function to set the event handler for the application. In this case, the `this` argument is specified.

This member function is available because the header file `ahellow3.hpp` includes the `icmdhdr.hpp` library header file, which contains the `ICommandHandler` class as follows:

```

#ifndef _AHELLOW3_
#define _AHELLOW3_
#include <iframe.hpp>
#include <istattxt.hpp>
#include <iinfoa.hpp>
#include <imenubar.hpp>
#include <icmdhdr.hpp>

```

### Add Command Processing

The next statements construct the command handler from a pointer to the `AHelloWindow` that events will be handled for. The `ahellow3.cpp` file uses the `ACommandHandler` to create a command handler, as follows:

```

ACommandHandler :: ACommandHandler(AHelloWindow *helloFrame)
 :frame(helloFrame)
{
}

```

Depending on the command event ID, you need to call the `AHelloWindow::setTextAlignment` function with the appropriate `AHelloWorld::Alignment` enumerator, as shown in the following sample from `ahellow3.cpp`:

```

bool
 ACommandHandler :: command(ICommandEvent & cmdEvent)
 {
 bool eventProcessed(true);
 switch (cmdEvent.commandId()) {
 case MI_CENTER:
 frame->setTextAlignment(AHelloWindow::center);
 break;
 case MI_LEFT:
 frame->setTextAlignment(AHelloWindow::left);
 break;
 case MI_RIGHT:
 frame->setTextAlignment(AHelloWindow::right);
 break;
 default:
 eventProcessed=false;
 }
 return(eventProcessed);
 }

```

The AHelloWorld::Alignment enumerator is defined in the ahellow3.hpp file:

```

class AHelloWindow : public IFrameWindow
{
 public:
 enum Alignment
 {
 left, center, right
 };
 // ...
};

```

#### RELATED CONCEPTS

- “Chapter 21. Resources” on page 297
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Convert Application Resources” on page 395
- “Add Menus to Your Application” on page 57
- “Add a Menu Bar” on page 59
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

IEvent  
 IHandler  
 IMenu

---

## Hello World Version 4: Adding Dialogs and Push Buttons

Version 4 modifies the menu bar and the pull-down menu in the following ways:

- Creates an **Edit** choice on the menu bar.
- Moves the **Alignment** choice from the menu bar to the pull-down menu.
- Moves the menu items associated with **Alignment** (**Left**, **Center**, and **Right**) from the pull-down menu into a cascaded menu that displays when **Alignment** is selected. These items still align the “Hello, World!!!” text string in the client



window. However, the commands assigned to these menu items are also assigned to accelerator keys so the keyboard can bypass the menu choices and establish the text alignment.

- Adds **Text** on the pull-down menu. Selecting this item displays a dialog box that contains an entry field in which the “Hello, World!!!” text string can be edited.

Hello World version 4 contains a resource file from the Open Class Library with definitions for accelerators.

#### OS/2

For OS/2 the accelerator definitions require that you specify the following line in your resource script (.rc) file:

```
#include <os2.h>
```

#### AIX

For AIX this include statement can stay in the resource file for portability without failure.

The main window for version 4 of the Hello World application looks like this:



#### List the Hello World Version 4 Files

You can find the following files in the samples/ioc/hello4 directory:

| File         | Type of Code                                                                        |
|--------------|-------------------------------------------------------------------------------------|
| adialog4.cpp | Source code to create the ATextDialog class                                         |
| adialog4.hpp | Header file for the ATextDialog class                                               |
| ahellow4.cpp | Source code for the main procedure, main window constructor, and command processing |
| ahellow4.hpp | Header file for the AHelloWindow class                                              |
| ahellow4.h   | Symbolic definitions for the hello4 executable file                                 |
| ahellow4.icc | This configuration file differs on each platform.                                   |

| File                      | Type of Code                                                                     |
|---------------------------|----------------------------------------------------------------------------------|
| ahellow4.ico              | Icon file for the hello4 executable file                                         |
| ahellow4.rc               | Resource file for the hello4 executable file                                     |
| ahellow4.icc and makefile | The configuration file and the makefile for building the hello4 executable file. |

### The Primary Source Code File

The ahellow4.cpp file contains the source code for the main procedure and the AHelloWindow and ACommandHandler classes. The tasks performed by this code are described in the following sections.

### The AHelloWindow Class Header File

The ahellow4.hpp file, like the ahellow3.hpp file, contains the class definition and interface specifications for the AHelloWindow class, with a few modifications for version 4.

### The Symbolic Definitions File

The ahellow4.h file contains the symbolic definitions for this application. These definitions provide the IDs for the application window components.

For version 4, the symbolic definition file contains the Hello World version 3 code, as well as the following:

- Three window IDs (WND\_TEXTDIALOG, WND\_MCCANVAS, WND\_STCANVAS) for ATextDialog.
- One window ID for the push button set canvas (WND\_BUTTONS). It also contains new string IDs (STR\_CENTERB, STR\_LEFTB, and STR\_RIGHTB) for the text strings used in the push buttons.
- Two menu IDs (MI\_EDIT and MI\_TEXT) for the menu bar **Edit** choice and the **Text** choice in the pull-down menu.
- Four definitions for the dialog window controls (DID\_OK, DID\_CANCEL, DID\_ENTRY, and DID\_STATIC).

### The Text Dialog Source Code File

The adialog4.cpp file contains the source code for the ATextDialog class constructor and functions created for version 4.

### The ATextDialog Class Header File

The adialog4.hpp file contains the class definition and interface specifications for the ATextDialog class.

### The Resource File

Version 4 provides a resource file, ahellow4.rc. This resource file assigns an icon and several text strings with the constants defined in the ahellow4.h file, the symbolic definitions File. It also contains resources for the menu bar and the accelerator keys. The ahellow4.h file is included in this resource file so the icon, text strings, and other resources can be associated with the appropriate IDs.

### OS/2

The file os2.h is included because it is the top level include file that includes all the files necessary for writing an OS/2 application.

The resource file for version 4 contains the version 3 code, as well as additional strings, updated menus, and command IDs. The first is the accelerator table of

command IDs assigned to the function keys. These command IDs are used in the cascaded menu to show the accelerator, or shortcut, key assignments. For example, with these assignments and the command processing in `ahellow4.cpp`, when users press the **F7** key, it is the same as if they select the **Left** choice in the cascaded menu.

VisualAge C++ provides tools for converting Windows, AIX, and OS/2-style resources. See "Convert Application Resources" on page 395 for more information about the resource file conversion tools.

### The Icon File

The `ahellow4.ico` file is used as the icon that displays when the application is minimized. This icon is the same as for versions 2 and 3.

VisualAge C++ provides tools for converting Windows, AIX, and OS/2 bitmaps and icons. See "Convert Application Resources" on page 395 for more information about the bitmap and icon conversion tools.

### The Configuration File and the Makefile

The files `ahellow4.icc` and `makefile` differ on each platform.

### Explore Hello World Version 4

The following sections describe each of the tasks performed by version 4 of the Hello World application that have not been described for previous versions.

### Add a Cascaded Menu to the Menu Bar

For version 4, there are several modifications to the menu bar and its associated pull-down menu.

Version 4 replaces **Alignment** on the menu bar with **Edit** and makes **Alignment** a menu item on the **Edit** pull-down menu. When the **Edit** pull-down menu displays, an arrow to the right of **Alignment** indicates that a cascaded menu will display to the right when it is selected.

> AIX

> OS/2

The **Alignment** and **Edit** choices are defined in the AIX and OS/2 `ahellow4.rc` file, as follows:

```
MENU WND_MAIN
 BEGIN
 SUBMENU "~Edit", MI_EDIT
 BEGIN
 SUBMENU "~Alignment", MI_ALIGNMENT
 BEGIN
 MENUITEM "~Left\tF7", MI_LEFT
 MENUITEM "~Center\tF8", MI_CENTER
 MENUITEM "~Right\tF9", MI_RIGHT
 END
 MENUITEM "~Text...", MI_TEXT
 END
 END
```

> WIN

This appears in the Windows `.rc` file as follows:

```

WND_MAIN MENUEX
 BEGIN
 POPUP "&Edit" , MI_EDIT
 BEGIN
 POPUP "&Alignment" , MI_ALIGNMENT
 BEGIN
 MENUITEM "&Left\tF7" , MI_LEFT
 MENUITEM "&Center\tF8" , MI_CENTER
 MENUITEM "&Right\tF9" , MI_RIGHT
 END
 MENUITEM "&Text..." , MI_TEXT
 END
 END
 END
END

```

All Windows menu resources need to use the menu keyword MENUEX (instead of MENU) when compiling menu resources. In addition, all submenus (which in Windows are specified in the resource file with the keyword POPUP) should be identified with a menu item identifier, just like in OS/2.

### Add Keyboard Accelerators

Keyboard accelerators are key sequences that perform the same actions as menu items. In version 3, the **Left**, **Center**, and **Right** choices appeared as items in a pull-down menu. In version 4, these choices become part of the cascaded menu and are assigned a function key. The menu items are defined in the in ahellow4.rc file, with text describing the function and the accelerator key to use. The following code, from the OS/2 resource file, shows the corresponding accelerator keys for each menu action:

```

MENUITEM "~Left\tF7", MI_LEFT
MENUITEM "~Center\tF8", MI_CENTER
MENUITEM "~Right\tF9", MI_RIGHT

```

The Windows menu items and accelerator keys are also defined in the in ahellow4.rc as follows:

```

MENUITEM "&Left\tF7" , MI_LEFT
MENUITEM "&Center\tF8" , MI_CENTER
MENUITEM "&Right\tF9" , MI_RIGHT

```

The Windows accelerators use an ampersand (&) instead of the tilde (~).

The \t indicates that the accelerator key name is tabbed to the right for readability.

This code conveys to users that the **Left**, **Center**, and **Right** alignment choices can be made by selecting a menu item with the mouse or keyboard, or they can use the keyboard function keys F7, F8, and F9.

The menu choices for **Left**, **Center**, and **Right** are visual indicators to the user that F7, F8, and F9 can be used. The main window in ahellow4.cpp must be programmed to use the accelerator. This is done by using the accelerator style on the AHelloWindow constructor, as shown in the following code from the ahellow4.cpp file:

```

AHelloWindow :: AHelloWindow(const unsigned long windowId)
 :IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon |
 IFrameWindow::accelerator,
 windowId)
 // ...
{
 // ...
}

```

The default processing for this style causes the resource file to be searched for an ACCELTABLE or ACCELERATORS definition for WND\_MAIN, which is the main window ID.

#### WIN

For Windows, the accelerator table for version 4 is defined in the ahellow4.rc file:

```
WND_MAIN ACCELERATORS
BEGIN
 VK_F7 , MI_LEFT , VIRTKEY
 VK_F8 , MI_CENTER , VIRTKEY
 VK_F9 , MI_RIGHT , VIRTKEY
END
```

#### AIX

#### OS/2

For OS/2 and AIX, the .rc file contains the following:

```
ACCELTABLE WND_MAIN
BEGIN
 VK_F7 , MI_LEFT , VIRTUALKEY
 VK_F8 , MI_CENTER , VIRTUALKEY
 VK_F9 , MI_RIGHT , VIRTUALKEY
END
```

### Add a Pull-Down Menu Choice

The final modification to the pull-down menu adds the **Text...** choice. By convention, the ellipsis (...) indicates that selecting this choice causes a dialog window to display.

The following code from the ahellow4.rc file adds the **Text** choice on the AIX or OS/2 operating system:

```
MENUITEM "Text...", MI_TEXT
```

The following figure shows the pull-down menu items and the cascaded menu.



### Add a Modal Dialog Window

A *dialog window* is a specific type of frame window containing window controls that gather information from the user. Typically, dialog windows are defined as

modal to the owner frame window, that is, the user must respond to the dialog window before returning to the previous frame.

#### OS/2

In OS/2 Presentation Manager (PM), you use dialog templates to define dialog windows externally to the application. The application creates an Open Class Library frame window with the same window ID defined in the resource file for the dialog template. Any controls from the dialog, for example entry fields, that you want to manipulate using the Open Class Library are also constructed from the corresponding control ID from the resource file. The following example shows an OS/2 dialog template that you could use with Hello World version 4:

```
DLGINCLUDE 1 "AHELLOW4.H"
DLGTEMPLATE WND_TEXTDIALOG LOADONCALL MOVEABLE DISCARDABLE
BEGIN
 DIALOG "Hello World Edit Dialog", WND_TEXTDIALOG, 17, 22, 137, 84,
 WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
 BEGIN
 DEFPUSHBUTTON "OK", DID_OK, 6, 4, 40, 14
 PUSHBUTTON "Cancel", DID_CANCEL, 49, 4, 40, 14
 LTEXT "Edit Text:", DID_STATIC, 8, 62, 69, 8
 ENTRYFIELD "", DID_ENTRY, 8, 44, 114, 8, ES_MARGIN
 END
END
```

If you use the Open Class Library only for AIX or if you write portable applications, you should not use dialog templates, because they are not supported in AIX. Instead, you can use Open Class Library canvas controls.

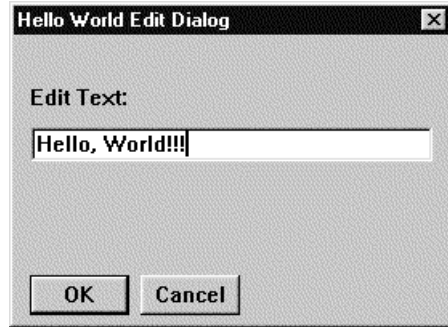
See “Chapter 13. Canvas Controls” on page 155 for information about the canvas classes. This Hello World version provides an example of using canvases.

In Hello World version 4, the dialog window contains the following:

- A prompt for the entry field. The prompt is a static text control with the text value loaded from the resource file.
- An entry field for changing the Hello World text string.
- An **OK** push button, indicating the change should be returned to the main window.
- A **Cancel** push button, indicating that the change should not be made.

Because these controls are best organized into columns and rows, use a multicell canvas control. However, because all controls in a column must be the same width, aligning the **OK** push button with the prompt would cause the **Cancel** push button to be placed to the right of the prompt text. Therefore, place the push buttons in a set canvas and align the set canvas as a single control with the prompt and entry field.

The dialog looks like this:



The following sections describe how to create, start, process, and end the dialog window.

### Invoke the Dialog Window

As mentioned in the previous section, the **Text** item on the pull-down menu causes a dialog window to display. It does this by calling the `AHelloWindow::editText` member function, as shown in the following code from the `ahellow4.cpp` file:

```
bool ACommandHandler :: command(ICommandEvent & cmdEvent)
{
 bool eventProcessed(true);
 switch (cmdEvent.commandId()) {
 case MI_CENTER:
 frame->setTextAlignment(AHelloWindow::center);
 break;
 case MI_LEFT:
 frame->setTextAlignment(AHelloWindow::left);
 break;
 case MI_RIGHT:
 frame->setTextAlignment(AHelloWindow::right);
 break;
 case MI_TEXT:
 frame->editText();
 break;
 default:
 eventProcessed=false;
 }
 return(eventProcessed);
}
```

The `editText` function saves the hello text string and sets the information area to indicate that the text dialog is active. The following code shows this:

```
AHelloWindow &
AHelloWindow :: editText()
{
 IString textValue(hello.text());
 infoArea.setDefaultText(STR_INFODLG);
 ATextDialog textDialog(textValue,this);
 textDialog.showModally();
 if (textDialog.result() == DID_OK)
 hello.setText(textValue);
 infoArea.setDefaultText(STR_INFO);
 return (*this);
}
```

Version 4 creates the `textDialog` data member from the `ATextDialog` class, a new class created as an `IFrameWindow` derived class. The following code from the `ahellow4.cpp` file defines the `editText` function:

```
ATextDialog textDialog(textValue,this);
```

The `textValue` object is passed as the current text string to the dialog.

### Construct the Dialog Window

When you create the `textDialog` object, the frame that represents the dialog and its components is initialized as shown in the following sample from the `adialog4.cpp` file:

```
ATextDialog :: ATextDialog(IString & textString, IWindow * ownerWnd)
 : IFrameWindow(IResourceId(WND_TEXTDIALOG)
 ,IWindow::desktopWindow()
 ,ownerWnd
 ,IRectangle()
 .moveBy(ownerWnd->rect().bottomLeft())
 ,IWindow::synchPaint
 |IWindow::clipSiblings
 |IWindow::saveBits
 |dialogBackground
 |dialogBorder
 |systemMenu
 |titleBar)
 ,clientCanvas(WND_MCCANVAS,this,this)
 ,buttons(WND_STCANVAS, &clientCanvas, &clientCanvas)
 ,statText(DID_STATIC,&clientCanvas,&clientCanvas)
 ,textField(DID_ENTRY,&clientCanvas,&clientCanvas)
 ,pushButton1(DID_OK,&buttons,&buttons)
 ,pushButton2(DID_CANCEL,&buttons,&buttons)
 ,dialogCommandHandler(this)
 ,saveText(textString)
{
 // ...
}
```

The `IFrameWindow` is initialized, in this case, with values that correspond to the dialog template. The `IRectangle` values approximate the size and position that would be generated from the dialog specifications. The style bits correspond to the bits that would be set by the template. Notice that dialog windows typically do not have sizing borders or minimize and maximize buttons.

Next, initialize the six controls needed for the dialog. Note that the buttons are owned by the set canvas; that the set canvas, the static text, and the entry field are owned by the multicell canvas; and that the multicell canvas is owned by the dialog window frame.

Because there are buttons in the frame, you must initialize a command handler for handling the **OK** and **Cancel** push buttons.

The `saveText` data member is initialized with the reference passed on the constructor. It provides the `setTextFromEntryField` function access to the edit string.

### Set the Dialog Window Controls

Once you initialize the dialog window controls, you must position and set them. The following code, from the `adialog4.cpp` file in the definition of the `ATextDialog` constructor, positions and sets the dialog window controls:

```
textField.setText(saveText);
textField.disableAutoScroll().enableMargin().enableTabStop();
statText.setText(DID_STATIC);
pushButton1.enableDefault().setText(IResourceId(DID_OK)).enableTabStop();
pushButton2.setText(IResourceId(DID_CANCEL));
buttons.setPackType(ISetCanvas::expanded).setMargin(ISize());
```



First, set the entry field. Then, set the initial text value from the saved text value.

You set the static text control's value from the resource file.

Set the push buttons by getting their text values from strings in the resource file and by enabling tabbing. Next, set `pushButton1` as the default push button, which means that when a user presses **Enter**, it is the same as pushing that button on the application.

You only need to turn on padding and set margins to zero for the set canvas containing the buttons. Turning on padding lets the buttons expand to fill the set canvas area.

Add the following controls to the multicell canvas:

- A static text field named `statText` at column 2, row 4
- An entry field named `textField` at column 2, row 6
- A set canvas of push buttons named `buttons` at column 2, row 14

The call to `setColumnWidth` horizontally separates the entry field and the right edge of the edit dialog by ten pixels. The call to `setRowHeight` vertically separates the set canvas of push buttons and the bottom edge of the edit dialog by five pixels:

```
clientCanvas.addToCell(&statText , 2, 4);
clientCanvas.addToCell(&textField, 2, 6);
clientCanvas.addToCell(&buttons, 2, 14);
clientCanvas.setColumnWidth(3, 10);
clientCanvas.setRowHeight(15, 5);
```

Finally, you position the multicell canvas in the dialog frame as the client, start the command handler for the dialog, and set the focus to the entry field. The following code, from the `adialog4.cpp` file, shows this:

```
setClient(&clientCanvas);
IRectangle dialogPosition(IPoint(0,0),
 clientCanvas.minimumSize());
dialogPosition.centerAt(
 IWindow::desktopWindow()->rect().centerXCenterY());
moveSizeToClient(dialogPosition);
dialogCommandHandler.handleEventsFor(this);
textField.setFocus();
```

### Process the Dialog Window

Once the `textDialog` has been created, the `editText` function displays and gives control to it using the `IFrameWindow::showModally` member function. Because the dialog window shows modally, it continues to have control until the `dismiss` function is called:

```
textDialog.showModally();
```

Once you give the dialog window control using the `IFrameWindow::showModally` function, the user can interact with the dialog in three ways:

- Because you set the focus to the entry field, the user's normal keystrokes are processed by the default edit handler and the entry field is edited.
- The user can use the system menu to close or move the dialog window. However, the window cannot be resized.
- The user can use the **Tab** keys or the mouse to select a push button. Also, because you defined the **OK** push button as a default key, users can use the **Enter** key to select it. When a user presses a push button, the

ADialogCommandHandler::command function processes the event. If the user presses **OK**, the setTextFromEntryField function is called to change the saved text value to the one edited in the text entry field. The code for this is found in the adialog4.cpp file, as follows:

```
ATextDialog &
ATextDialog::setTextFromEntryField()
{
 saveText = textField.text();
 return (*this);
}
```

The dismiss function is then called with the DID\_OK ID. This value is saved in the textDialog's IFrameWindow object. The dismiss function closes the window and returns control to the owner window. This is found in the adialog4.cpp file, as follows:

```
bool ADialogCommandHandler :: command(ICommandEvent & cmdEvent)
{
 bool eventProcessed(true);
 switch (cmdEvent.commandId()) {
 case DID_OK:
 frame->setTextFromEntryField();
 frame->dismiss(DID_OK);
 break;
 case DID_CANCEL:
 frame->dismiss(DID_CANCEL);
 break;
 default:
 // The event was not processed
 eventProcessed=false;
 }
 return(eventProcessed);
}
```

Pressing the **Cancel** push button does not call the setTextFromEntryField function, but it does call the IFrameWindow::dismiss function with the DID\_CANCEL value. This is found in the adialog4.cpp file, as follows:

```
case DID_CANCEL:
 frame->dismiss(DID_CANCEL);
 break;
```

The AHelloWindow::editText function can then use the IFrameWindow::result function to determine if the user changed the text value. The information area is also reset. The following code is from the ahellow4.cpp file:

```
AHelloWindow &
AHelloWindow :: editText()
{
 IString textValue(hello.text());
 infoArea.setDefaultText(STR_INFODLG);
 ATextDialog textDialog(textValue,this);
 textDialog.showModally();
 if (textDialog.result() == DID_OK)
 hello.setText(textValue);
 infoArea.setDefaultText(STR_INFO);
 return (*this);
}
```

The code for the text dialog comes from the adialog4.cpp file. The declaration and interface specifications for the ATextDialog class are contained in the adialog4.hpp file, which is included by both the ahellow4.cpp and adialog4.cpp files.

## Delete the Dialog Window

Because you create the `textDialog` object statically, that is, the `new` operator is not used to create it, the object is deleted automatically when it is no longer in scope, in this case, when the `editText` function is exited. Using this approach means that each time the `editText` function is called, a new `textDialog` object is created, processed, and deleted.

## Set Push Buttons in a Set Canvas

The following code, from the `ahellow4.hpp` file, defines the `buttons` data member as an instance of the `ISetCanvas` class:

```
ISetCanvas
 buttons;
```

To make the `ISetCanvas` class available to the application, the `ahellow4.hpp` file includes the `isetcv.hpp` library header file, as follows:

```
#include <isetcv.hpp>
```

Next, the `buttons` data member is created as a set canvas control with the main window as the parent and owner of the control. The `WND_BUTTONS` constant provides the window ID for this set canvas control. The following is in `ahellow4.cpp` file:

```
AHelloWindow :: AHelloWindow(const unsigned long windowId)
 :IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon |
 IFrameWindow::accelerator,
 windowId)
 ,menuBar(windowId, this)
 ,statusLine(WND_STATUS, this, this)
 ,hello(WND_HELLO, this, this)
 ,buttons(WND_BUTTONS, this, this)
 ,leftButton(MI_LEFT, &buttons, &buttons)
 ,centerButton(MI_CENTER, &buttons, &buttons)
 ,rightButton(MI_RIGHT, &buttons, &buttons)
 ,infoArea(this)
 ,commandHandler(this)
{
 // ...
}
```

The `WND_BUTTONS` constant is defined in `ahellow4.h` as follows:

```
#define WND_BUTTONS 0x1070
```

Use the `setMargin` and `setPad` member functions to set the canvas margins and pad to zero. The following code, from the `ahellow4.cpp` file, shows how to do this:

```
leftButton.enableTabStop();
buttons.setMargin(ISize());
buttons.setPad(ISize());
addExtension(&buttons, IFrameWindow::belowClient,
 (unsigned long)buttons.minimumSize().height());
```

## Define the Push Buttons

Now that you have a set canvas, define two push button data members in the header file, `adialog4.hpp`, as shown in the following code:

```
class ATextDialog : public IFrameWindow
{
 // ...
private:
 IPushButton
```

```

 pushButton1,
 pushButton2;
 // ...
};

```

### Create Push Buttons

The `adialog4.hpp` file includes the `ipushbut.hpp` library header file and makes the `IPushButton` class available to version 4. You need the data members defined in the `ahellow4.hpp` file to create three push buttons in the set canvas: **Left**, **Center**, and **Right**. The `ahellow4.hpp` file uses the following code to include the `ipushbut.hpp` file:

```
#include <ipushbut.hpp>
```

The following code creates a new instance of the **Left** push button control and specifies that it uses the command processing associated with the `MI_LEFT` menu item attribute to align the "Hello, World!!!" text string on the left side of the client window. The following code comes from `ahellow4.cpp`:

```
,leftButton(MI_LEFT, &buttons, &buttons)
```

Other than the data member used (`centerButton` is used for the **Center** push button and `rightButton` is used for the **Right** push button), the window ID is the only difference in the code that is used to create all three push buttons. Specify the `MI_CENTER` menu item window ID for the **Center** push button and `MI_RIGHT` for the **Right** push button.

The set canvas control is identified as the owner and parent of the push button control.

### Set Text in Push Buttons

The `ahellow4.cpp` file uses the `setText` member function to set text strings in each push button. Here is the code that sets the text in the **Left** push button:

```

AHelloWindow :: AHelloWindow(const unsigned long windowId)
 :IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon |
 IFrameWindow::accelerator,
 windowId)
 ,menuBar(windowId, this)
 ,statusLine(WND_STATUS, this, this)
 ,hello(WND_HELLO, this, this)
 ,buttons(WND_BUTTONS, this, this)
 ,leftButton(MI_LEFT, &buttons, &buttons)
 ,centerButton(MI_CENTER, &buttons, &buttons)
 ,rightButton(MI_RIGHT, &buttons, &buttons)
 ,infoArea(this)
 ,commandHandler(this)
{
 // ...
}

```

Other than the data member for which the text is set (`centerButton` is used for the **Center** push button and `rightButton` is used for the **Right** push button), the only difference between this code and the code that puts text in the other two push buttons is the `STR_LEFTB` constant, which associates with the appropriate text string in the resource (`ahellow4.rc` or `.wrc`) file. Here are the text string associations for all three push buttons:

```

STR_LEFTB , "Left"
STR_CENTERB , "Center"
STR_RIGHTB , "Right"

```

#### RELATED CONCEPTS

- “Chapter 21. Resources” on page 297
- “Chapter 13. Canvas Controls” on page 155
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

- “Convert Application Resources” on page 395
- “Chapter 17. Common Dialogs” on page 261
- “Use Keyboard Accelerators” on page 50
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

#### RELATED REFERENCES

- ICanvas
- IResource

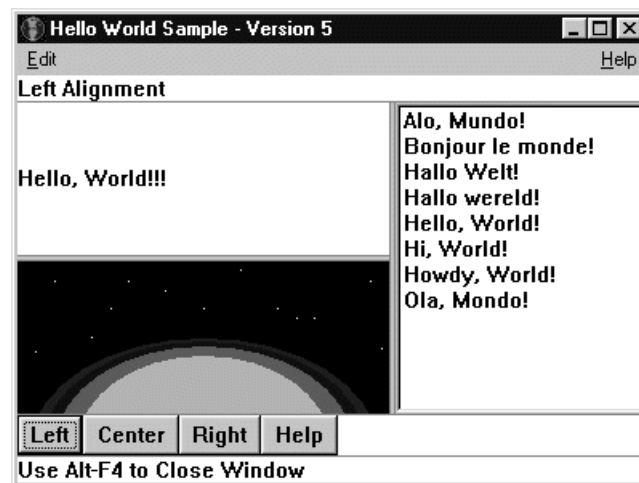
---

## Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help

Version 5 of the Hello World application shows you how to add the following:

- Split canvases for showing multiple control windows in the client window
- List box for selecting text for a static text window
- Help window to provide users with information about the frame window
- 2D-graphics and drawing canvases

The main window for version 5 of the Hello World application looks like this:



### List the Hello World Version 5 Files

You can find the following files in the samples/ioc/hello5 directory:

| File         | Type of Code                                          |
|--------------|-------------------------------------------------------|
| ahellow5.cpp | Source code for main procedure and AHelloWindow class |
| ahellow5.hpp | Class header file for AHelloWindow                    |

| File                         | Type of Code                                                                    |
|------------------------------|---------------------------------------------------------------------------------|
| ahellow5.h                   | Symbolic definitions file shared by all source files                            |
| adialog5.cpp                 | Source code for the ATextDialog class                                           |
| adialog5.hpp                 | Class header file for ATextDialog                                               |
| aearthw5.cpp                 | Source code for the AEarthWindow class                                          |
| aearthw5.hpp                 | Class header file for AEarthWindow                                              |
| ahellow5.rc                  | Resource file source                                                            |
| ahellow5.ico                 | Icon file                                                                       |
| ahellow6.rtf and ahello6.hpj | Help source files for Windows help                                              |
| ahello6.ipf                  | IPF help source files                                                           |
| ahellow5.icc and makefile    | The configuration file and the makefile for building the hello5 executable file |

### **The Primary Source Code File**

The ahellow5.cpp file contains the source code for the main procedure and AHelloWindow class functions. The tasks performed by this code are described in the following sections.

### **The AHelloWindow Class Header File**

The ahellow5.hpp file contains the class definition and interface specifications for the AHelloWindow class, with a few modifications for version 5.

### **The Symbolic Definitions File**

The ahellow5.h file contains the symbolic definitions for this application. These symbols and their definitions provide the IDs for the application window components.

### **The Text Dialog Source Code File**

The adialog5.cpp file contains the source code for the ATextDialog class, modified for help in version 5.

### **The ATextDialog Class Header File**

The adialog5.hpp file contains the class definition and interface specifications for the ATextDialog class. The adialog5.hpp file is the same as the adialog4.cpp file.

### **The Earth Window Source File**

The aearthw5.cpp file contains the source code for the Earth window graphic that is drawn using native-system graphics calls.

### **The AEarthWindow Class Header File**

The aearthw5.hpp file contains the class definition and interface specifications for the AEarthWindow class.

### **The Resource File**

Version 5 of the Hello World application provides a resource file, ahellow5.rc, which contains all the resources from version 4, as well as additional resources, including a help table.

### **The Icon File**

The ahellow5.ico file is used as the icon that displays when the application is minimized. This icon is the same as for previous versions of Hello World.

## The Help Window Source Files

The ahellow5.ipf file contains text with IPF tags. The ahellow5.rtf and ahellow5.hpj files contain text with RTF tags. The Hello World application uses these files to generate the help information. IPF uses a tag language to format the text that appears in a help window. For example, :p. is the paragraph tag, which you use to start a new paragraph.

### OS/2

The OS/2 makefiles provided with Hello World version 5 use the IPFC compiler, provided by the OS/2 Developer's Toolkit, to compile the help file.

Refer to the *OS/2 Information Presentation Facility Guide and Reference* or the *IPF Information Presentation Facility User and Programmer's Guide* for descriptions of other tags used in the help source file.

### AIX

The Motif makefiles provided with Hello World version 5 use the IPFC compiler as well.

### WIN

The makefiles provided with Hello World version 5 use the native Windows help compiler, by default, which uses Rich Text Format (RTF). You can also use IPFC/X compiler to compile the help file by specifying the following line in your makefile:

```
nmake USE_IPF=1
```

A sample file, ahellow5.rtf is provided with the Hello World version 5. You also need a project file (.HPJ) which is used by the Windows Help Compiler (HCW or HC31) to create the .HLP file.

## The Configuration File and the Makefile

The files ahellow5.icc and makefile differ on each platform.

## Explore Hello World Version 5

The following sections describe each of the tasks performed by version 5 of the Hello World application that were not described for previous versions.

### Construct the Client Window with Split Canvases

In previous versions of the Hello World application, the client window contained a simple static text window. Version 5 provides a sample of a client window with three visible windows:

- A static text window containing "Hello World!!!" from the previous versions
- A new static text window with a graphical view of the earth from space
- A new list box containing different language versions of the phrase, "Hello World!!!"

The two new windows are described in detail in later sections of this chapter.

Use the ICanvas class for a common and easy-to-use method for placing multiple windows into a client window. Version 5 places the two static text windows into a horizontal split canvas, called helloCanvas, and then places helloCanvas and the list box into a vertical canvas called clientWindow. It also places windows in a split canvas by identifying the canvas as the control window's parent. For example, the

split canvases in the client area of the Hello World main window are initialized in the AHelloWindow constructor in the ahellow5.cpp file, as follows:

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
: IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon |
 IFrameWindow::accelerator,
 windowId)
,menuBar(windowId, this)
,clientWindow(WND_CANVAS, this, this)
,helloCanvas(WND_HCANVAS,
 &clientWindow,
 &clientWindow,
 IRectangle(),
 IWindow::visible |
 ISplitCanvas::horizontal)
,hello(WND_HELLO,
 &helloCanvas,
 &helloCanvas)
,statusLine(WND_STATUS, this, this)
,earthWindow(WND_EARTH, &helloCanvas)
,listBox(WND_LISTBOX,
 &clientWindow,
 &clientWindow,
 IRectangle(),
 IListBox::defaultStyle() |
 IControl::tabStop |
 IListBox::noAdjustPosition)
,buttons(WND_BUTTONS, this, this)
,leftButton(MI_LEFT, &buttons, &buttons)
,centerButton(MI_CENTER, &buttons, &buttons)
,rightButton(MI_RIGHT, &buttons, &buttons)
,helpButton(MI_HELP,
 &buttons,
 &buttons, IRectangle(),
 IPushButton::defaultStyle() |
 IPushButton::help |
 IButton::noPointerFocus)
,infoArea(this)
,commandHandler(this)
,selectHandler(this)
,helpWindow(HELP_TABLE, this)
{
// ...
}
```

The order in which you place your child windows into a split canvas determines the order in which they will be seen. For example, the AHelloWindow::hello static text window appears above the earthWindow static text window because it is created as the first child of the helloCanvas window. Likewise, the helloCanvas window appears to the left of the list box because it is created before the list box.

After initializing the canvases, the following statements from the ahellow5.cpp file set the client window and the proportions for its child windows in the vertical split canvas.

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
// ...
{
// ...
setClient(&clientWindow);
clientWindow.setSplitWindowPercentage(&helloCanvas, 60);
clientWindow.setSplitWindowPercentage(&listBox, 40);
// ...
}
```



## Create and Use a List Box

Hello World version 5 provides you with the ability to change the Hello World text by selecting different language versions of the phrase "Hello World" from a list box. The `IListBox` object in Hello World version 5 is called `listBox` and is initialized in the `AHelloWindow` constructor in `ahellow5.cpp` as follows:

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
// ...
, listBox(WND_LISTBOX,
 &clientWindow,
 &clientWindow,
 IRectangle(),
 IListBox::defaultStyle() |
 IControl::tabStop |
 IListBox::noAdjustPosition)
// ...
{
// ...
}
```

The inherited `tabStop` style indicates that you can tab to the list box. The `noAdjustPosition` style prevents the list box from being automatically resized when an item in the list does not fit inside the current window.

One way you can populate the newly created list box is to use `IListBox::addAscending`. This function adds a text string to the list box in ascending alphabetical order. For example, Hello World version 5 uses the `addAscending` function to load a variable number of strings from the resource file into the list box as follows:

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
// ...
{
// ...
for (int i=0;i<HI_COUNT;i++)
listBox.addAscending(HI_WORLD+i);
// ...
}
```

The strings are defined in the resource file, `ahellow5.rc`, as follows:

```
// Change HI_COUNT in ahellow5.h to change number of HIs used.
HI_WORLD, "Hello, World!" //English
HI_WORLD+1, "Hi, World!" //American
HI_WORLD+2, "Howdy, World!" //Southern American
HI_WORLD+3, "Alo, Mundo!" //Portuguese
HI_WORLD+4, "Ola, Mondo!" //Spanish
HI_WORLD+5, "Hallo wereId!" //Dutch
HI_WORLD+6, "Hallo Welt!" //German
HI_WORLD+7, "Bonjour le monde!" //French
HI_WORLD+8, "Put your language here!" //Add more items, too!
```

`HI_WORLD` is a symbolic definition for the constant string ID of the first string. `HI_COUNT` is the symbolic definition for the constant number of strings to load. In this sample, `HI_COUNT` is defined as 8. Therefore, only `HI_WORLD` through `HI_WORLD+7` are loaded. You can add another item to the list by changing the `HI_WORLD` constant in the symbolic definition file, `ahellow5.h`, and adding your strings to the `HI_WORLD` list.

Once you create the list box and fill it with items, define a select handler for processing list box selections. Hello World version 5 provides a select handler class, called `ASelectHandler`. (It is similar to the command handler added in Hello World version 3.)

The three differences are as follows:

- The handler function that you override is called `ISelectHandler::selected` instead of `ICommandHandler::command`.
- The event passed into the function is an `IControlEvent`, instead of an `ICommandEvent`.
- The function that is being called processes the selections that have been made to our list box.

The `ASelectHandler` is defined in the `AHelloWindow` class header file, `ahellow5.hpp`, using the following class definition:

```
class ASelectHandler : public ISelectHandler
{
public:
 ASelectHandler(AHelloWindow *helloFrame);
 virtual ~ASelectHandler() { }
protected:
 virtual bool selected(IControlEvent& ctlEvent);
private:
 AHelloWindow *frame;
};
```

The `ASelectHandler::selected` function and the `AHelloWindow::setTextFromListBox` function provide the selection event handling for version 5. These functions are listed below and can be found in the `ahellow5.cpp` file:

```
AHelloWindow &
AHelloWindow :: setTextFromListBox()
{
 IListBox::Cursor lbCursor(listBox);
 lbCursor.setToFirst();
 hello.setText(listBox.elementAt(lbCursor));
 return (*this);
}
// ...
bool ASelectHandler :: selected(IControlEvent & evt)
{
 frame->setTextFromListBox();
 return (true);
}
```

The `setTextFromListBox` function uses the typedef `ILListBox::Cursor`. Use a list box cursor to scan through the items in a list box. The constructor used in Hello World version 5 for creating the `lbCursor` object contains only one argument, the list box object to be scanned. You can also specify an additional argument, called a *filter*, if you want to scan all the items in the list or only the items that are selected. Because the `setTextFromListBox` function looks for the first item selected it uses the default filter type. The default filter type is *selected*.

The `ILListBox::Cursor::setToFirst` function positions the cursor to the first selected item. Then, the `ILListBox::elementAt` function uses the cursor to locate and return the text string identified by the first selected item. Hello World version 5 uses the string value to set the Hello World text.

Like the command handler in Hello World version 3, the `ASelectHandler::selected` function is not called until you attach the handler to the proper window. In this case, you attach the select handler to the list box using the `ASelectHandler` inherited `IHandler` function, `handleEventsFor`. This causes the `ASelectHandler::selected` function to be called each time you select a list box item.

To stop handling selection events, use the `stopHandlingEventsFor` function, again specifying the list box.

### Use Native System Functions and Graphics

The `aearthw5.hpp` and `aearthw5.cpp` files contain the class header and implementation for the `AEarthWindow` class. Refer to the comments and code in these files for examples of the following:

- Displaying a graphics window using `IDrawingCanvas`, `IGEllipse2D`, and `IGrafBundle` objects
- Creating and using an `IResizeHandler` object to resize the window whenever it is changed
- Representing geometric shapes using `IRectangle` and `IPoint` objects
- Using `IColor` objects as arguments to native function calls

### Set Up the Help Area

Use the following steps to create IPF help information for your application:

1. Create a file containing the help information.

Create the source text that displays in your application's help window using the IPF format (`.ipf` file) for the OS/2 operating system and the IPF/X format (`.ipf` file) for the Windows and Motif environments. Compile your file into a help file (`.hlp` file) using the IPFC compiler.

Refer to the Information Presentation Facility documentation for descriptions of the tags you use to create the source `.ipf` file.

For an example of an `.ipf` source file, see the Hello World version 5 `ahellow5.ipf` file.

2. Define the help window title and the help submenu in your resource script file.

#### WIN

The following is from the Windows `ahellow5.rc` file:

```
STR_HTITLE, "C++ Hello World - Help Window" //Help window title string
POPUP "&Help" , MI_HELP , RIGHTJUSTIFY
BEGIN
 MENUITEM "&General help..." , SC_HELPEXTENDED /*MIS_SYSCOMMAND*/
 MENUITEM "&Keys help..." , SC_HELPKEYS /*MIS_SYSCOMMAND*/
 MENUITEM "Help &index..." , SC_HELPINDEX /*MIS_SYSCOMMAND*/
END
```

#### AIX

#### OS/2

The following is from the AIX or OS/2 resource script file:

```
STR_HTITLE, "C++ Hello World - Help Window" //Help window title string
SUBMENU "~Help", MI_HELP, MIS_HELP //Help submenu
BEGIN
 MENUITEM "~General help...", SC_HELPEXTENDED, MIS_SYSCOMMAND
 MENUITEM "~Keys help...", SC_HELPKEYS, MIS_SYSCOMMAND
 MENUITEM "Help `index...", SC_HELPINDEX, MIS_SYSCOMMAND
END
```

`MI_HELP` is the help menu ID.

Normally, you specify `MIS_HELP` for a menu item to cause a help event, rather than a command event, to be posted when the menu item is selected.

When you specify MIS\_HELP (OS/2) or RIGHTJUSTIFY (Windows) for a submenu item, Motif automatically positions the item to the far right of the menu bar. OS/2 PM ignores MIS\_HELP specified on submenu items.

When MIS\_SYSCOMMAND is specified with the predefined SC\_HELP\* IDs, a system command event is generated. The default system command handler recognizes the predefined IDs and shows the appropriate help panel, except for SC\_HELPKEYS, which by default does nothing. SC\_HELP\* IDs are defined in the iconst.h file. You can override this default processing for SC\_HELPKEYS, using an IHelpHandler, which is described in a later step.

### 3. Define a help table in the resource file.

The help table defines the relationship between the window ID and the general or contextual panel ID that is defined in the .ipf file. The following help table is defined in the resource file, ahellow5.rc, for Hello World version 5:

```
HELPTABLE HELP_TABLE
BEGIN
 HELPITEM WND_MAIN, SUBTABLE_MAIN, 100
 HELPITEM WND_TEXTDIALOG, SUBTABLE_DIALOG, 200
END
HELPSUBTABLE SUBTABLE_MAIN //Main window help subtable
BEGIN //
 HELPSUBITEM WND_HELLO, 100 //Hello static text help ID
 HELPSUBITEM WND_LISTBOX,102 //List box help ID
 HELPSUBITEM MI_EDIT, 110 //Edit menu item help ID
 HELPSUBITEM MI_ALIGNMENT, 111 //Alignment menu item help ID
 HELPSUBITEM MI_LEFT, 112 //Left command help ID
 HELPSUBITEM MI_CENTER, 113 //Center command help ID
 HELPSUBITEM MI_RIGHT, 114 //Right command help ID
 HELPSUBITEM MI_TEXT, 199 //Text command help ID
END
HELPSUBTABLE SUBTABLE_DIALOG //Text dialog help subtable
BEGIN //
 HELPSUBITEM DID_ENTRY, 201 //Entry field help ID
 HELPSUBITEM DID_OK, 202 //OK command help ID
 HELPSUBITEM DID_CANCEL, 203 //Cancel command help ID
END
```

WND\_HELLO and WND\_LISTBOX are control IDs, MI\_\* are menu item IDs, and the DID\_\* are push button IDs. Each window ID is related to a help panel ID. In the preceding example, WND\_MAIN and WND\_HELLO both correspond to help panel ID 100. That is, pressing the **F1** key in the main window area displays the same help panel as selecting **General help** from the **Help** menu.

### 4. Create a help window object for your application window.

Use the IHelpWindow class to associate help information with an application window. Hello World version 5 defines the private data member, helpWindow, as an IHelpWindow object. It is initialized in the AHelloWindow constructor in ahellow5.cpp using the following statement:

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
: IFrameWindow(IFrameWindow::defaultStyle() |
 IFrameWindow::minimizedIcon |
 IFrameWindow::accelerator,
 windowId)
// ...
,helpWindow(HELP_TABLE,this)
{
// ...
}
```

The first parameter identifies the help table defined in the resource file. The second parameter identifies the application window for which the help is being provided.

Use the `IHelpWindow::addLibraries` member function to associate a help file with a help window. The `AHelloWindow` constructor in Hello World version 5 provides an example:

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
// ...
{
// ...
try
{
 helpWindow.addLibraries("ahellow5.hlp");
 helpWindow.setTitle(STR_HTITLE);
 helpHandler.handleEventsFor(this);
}
catch(...)
{
 MessageBox msgBox(this);
 msgBox.show(STR_HELP_NOT_FOUND, MessageBox::warning);
}
// ...
}
```

You can set the title of the help window by using the `IHelpWindow::setTitle` member function. The following code sets the title from a string defined in the resource file:

```
helpWindow.setTitle(STR_HTITLE);
```

5. Create a help handler by creating an `IHelpHandler` derived class.

A help handler processes help events. Create your own help handler class derived from `IHelpHandler` to provide help event processing that is unique to your application. Hello World version 5 uses a help handler only to display the keys help panel describing accelerator key definitions.

To override keys help processing, do the following:

- a. Define a new class that is derived from `IHelpHandler`, as shown in the Hello World version 5 class definition file, `ahellow5.hpp`.

```
class AHelpHandler : public IHelpHandler
{
public:
 virtual ~AHelpHandler() { }
protected:
 virtual bool keysHelpId(IEvent& evt);
};
```

- b. Provide the overridden virtual function `keysHelpId`, which is called when keys help is requested. The following code, from the Hello World version 5 `ahellow5.cpp` file, shows how to implement this function.

```
bool
AHelpHandler :: keysHelpId(IEvent& evt)
{
 // 1000 = keys help ID in ahellow5.ipf file
 evt.setResult(1000);
 return (true);
}
```

In the preceding code, the help panel ID for the Hello World version 5 keys help is set in the event result.

- c. Start and stop help event processing.

Your help handler, previously described, does not begin handling help events until you use the `handleEventsFor` member function. For example, the following code causes the `helpHandler` to begin processing help events for this frame window:

```
helpHandler.handleEventsFor(this)
```

Typically, you include this statement in the constructor for the frame window.

Note that the window that handles help events must be an associated window. That is, you should identify the window as the associated window on the `IHelpWindow` constructor or explicitly identify the window as an associated window using the `IHelpWindow::setAssociatedWindow` function.

When you want to stop handling help events, for example, when you close your frame window, use the `stopHandlingEventsFor` member function, as follows:

```
helpHandler.stopHandlingEventsFor(this);
```

You typically include this statement in the destructor for the frame window.

### **Assign Help IDs to Windows**

You can assign helpIDs directly to windows by using the `IWindow::setHelpId` function while maintaining portability. When **F1** is pressed, the window with focus is then queried for its context helpID and if one is found, it is used to identify the help panel to display. If one is not found and there is a help table, it is searched for the appropriate helpID to use.

If you are not using a help table, the `helpId` of the active frame is used as the general help panel.

### **Use Native Windows Help Files with IHelpWindow**

By default, `IHelpWindow` uses the native help for the platform to handle help requests using a `.hlp` file that was created from RTF or IPF source. This allows the code you write to be portable while giving your users native look-and-feel. IPF help files are also portable between the various platforms. RTF help files are not portable. In addition, some functions provided by IPF are not available when using Windows native help (RTF). The major differences are as follows:

- There is only one instance of Windows Help for all your applications.
- The size and position of WinHelp is not relative to any particular application.
- IPF uses a set of file names to locate help; WinHelp is one file at a time, but you must still use `IHelpWindow::addLibraries` to change the help file name.
- IPF help has sections for CONTENTS, INDEX, KEYS, GENERAL, and USING. WinHelp has HELP\_FINDER and HELPONHELP.
- Only IPF allows the application to change the Help Manager menu and accelerators.
- Only IPF allows the application to set a title for each help object.
- Only IPF allows querying of handles for IPF sections, such as the cover page or contents.
- Only IPF allows the application to toggle showing of the help IDs in the help manager window titles.

#### **RELATED CONCEPTS**

- “Chapter 13. Canvas Controls” on page 155
- “Chapter 21. Resources” on page 297
- “Chapter 1. Windows” on page 1
- “Chapter 3. Events and Event Handlers” on page 19

**RELATED TASKS**

- “Convert Application Resources” on page 395
- “Add Menus to Your Application” on page 57
- “Create a Split Canvas” on page 165
- “Create a Push Button” on page 99
- “Create a List Box” on page 119
- “Write an Event Handler” on page 25
- “Task and Samples Cross-Reference Table” on page 445

**RELATED REFERENCES**

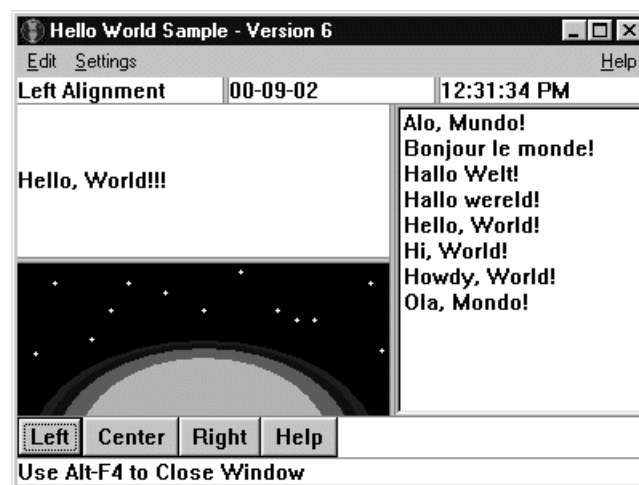
- IEvent
- IMenu
- IMenuBar

## Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics

Version 6 of the Hello World application shows you how to do the following:

- Use a font dialog to change the font for a static text window
- Use a pop-up menu and a menu handler
- Use a notebook with multiple controls for changing application settings
- Use the IProfile class to save and read user settings
- Use a combination box instead of an entry field
- Add help to your application using native Windows help (RTF) or the OS/2 Information Presentation Facility (IPF) for portable help files
- Use the 2D graphics classes

The main window for Version 6 of the Hello World application looks like this:



### List the Hello World Version 6 Files

The following files contain the code used to create version 6:

| File                        | Type of Code                                                                |
|-----------------------------|-----------------------------------------------------------------------------|
| ahello6.cpp                 | Source code for main procedure and AHelloWindow class                       |
| ahello6.hpp                 | Class header file for AHelloWindow                                          |
| ahello6.h                   | Symbolic definitions file shared by all source files                        |
| adialog6.cpp                | Source code for the ATextDialog class                                       |
| adialog6.hpp                | Class header file for ATextDialog                                           |
| aearthw6.cpp                | Source code for the AEarthWindow class                                      |
| aearthw6.hpp                | Class header file for AEarthWindow                                          |
| anotebw6.cpp                | Source code for the ANotebookWindow class                                   |
| anotebw6.hpp                | Class header file for ANotebookWindow                                       |
| ahello6.rc                  | Resource file                                                               |
| ahello6.ico                 | Icon file                                                                   |
| ahello6.rtf and ahello6.hpj | Help source files for Windows help                                          |
| ahello6.ipf                 | IPF help source files                                                       |
| ahello6.icc and makefile    | The configuration file and makefile for building the hello6 executable file |

### Explore Hello World Version 6

The following list describes the tasks performed by version 6 of the Hello World application that are not already described for previous versions. The tasks are as follows:

- Implementing an `ITimer` object for updating the time and the twinkling of the stars  
Refer to `timehdr.cpp` and `timehdr.hpp` in the `ownhdr` sample for an example of writing your own handler. This sample demonstrates writing your own handler by implementing a simple time handler that posts a `WM_TIMER` event and calls `ATimeHandler::tick` every second.  
The `ownhdr` sample demonstrates `IHandler` derivation; the timer functions might not handle all cases and might break in a multithreaded environment.  
Refer to the `aearthw6.cpp` and `aearthw6.hpp` files for `ATwinkleTimeHandler`. The `ATwinkleTimeHandler` uses the `ITimer` class to provide stars that twinkle.
- Changing the status line in `AHelloWindow` to a split canvas with the status alignment plus the current date and time  
Hello World version 6 also provides public functions for setting the date and time formats.
- Adding a pop-up menu and a menu handler to the `AHelloWindow` class  
Hello World version 6 derives a new class, `APopUpHandler`, from `IMenuHandler`, and overrides the virtual function `makePopupMenu` to provide pop-up menus for the `hello` and `earthWindow` static text windows. This version also demonstrates using static and dynamic pop-up menus.
- Adding a new **Edit** menu item, **Fonts**  
This new menu item invokes the `AHelloWindow::setHelloFont` function that uses a modal `IFontDialog` to change the font in the `AHelloWindow::hello` static text window.
- Adding a new submenu to the **Settings** menu item



This new submenu contains **Read**, **Open**, and **Save** menu items.

- Adding a new Settings menu item, **Open**

This new menu item invokes the `AHelloWindow::openHelloSettings` function that dynamically creates a nonmodal `ANotebookWindow` frame window. The nonmodal frame window lets you change `AEarthWindow` settings and date and time formats using check boxes, a slider, and radio buttons from a notebook window.

Refer to `anotebw6.hpp` and `anotebw6.cpp` to see how the new `ANotebookWindow` class is defined and implemented, and to `ahellow6.cpp` for an example of its use.

- Adding a new **Settings** menu item, **Save**

This new menu item invokes the `AHelloWindow::saveHelloSettings` function that saves the hello font and the changeable user settings to an `IProfile` class file. Hello World version 6 uses a message box to indicate that the save completed successfully.

- Adding a new **Settings** menu item, **Read**

This new menu item invokes the `AHelloWindow::readHelloSettings` function that reads the hello font and the changeable user settings from an `IProfile` class file. Hello World version 6 uses a message box to indicate that the read completed successfully.

- Enhancing the `AEarthWindow` function using graphics to include the following:
  - Setting and querying the intensity of the stars
  - Making the stars start and stop twinkling
  - Setting and querying the number of atmosphere layers
  - Setting and querying the color of the earth

Refer to the `aearthw6.cpp` and `aearthw6.hpp` files for more information.

#### RELATED CONCEPTS

“Chapter 21. Resources” on page 297

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

#### RELATED TASKS

“Convert Application Resources” on page 395

“Add Menus to Your Application” on page 57

“Create a Split Canvas” on page 165

“Create a Push Button” on page 99

“Create a List Box” on page 119

“Use Notebooks” on page 204

“Chapter 17. Common Dialogs” on page 261

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table”

---

## Task and Samples Cross-Reference Table

The following table contains a list of some Open Class Library tasks and cross-references them to the samples and examples used in the information that show you how to complete the task. You can find all the samples in the `samples/ioc` directory:

| Tasks                                                                                   | Sample or Example                                                                                                                                                                                                                                                                                                                                                                                                                                       | Class Usage                  |
|-----------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| Create a basic frame window with a line of text centered in the middle of the window    | "Hello World Version 1: Create a Main Window" on page 402                                                                                                                                                                                                                                                                                                                                                                                               | IFrameWindow<br>IStaticText  |
| Align a text string within a static text control                                        | "Hello World Version 2: Add Resource Files and Frame Extensions" on page 405<br><br>"Create a Split Canvas" on page 165                                                                                                                                                                                                                                                                                                                                 | IStaticText                  |
| Display information about an application in an information area below the client window | "Hello World Version 2: Add Resource Files and Frame Extensions" on page 405<br><br>"Hello World Version 3: Add Command Handlers and Menu Bars" on page 412<br><br>"Hello World Version 4: Adding Dialogs and Push Buttons" on page 420<br><br>"Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help" on page 433<br><br>"Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics" on page 443 | IFrameExtension<br>IInfoArea |
| Title your application using an .rc file                                                | "Hello World Version 2: Add Resource Files and Frame Extensions" on page 405<br><br>"Hello World Version 3: Add Command Handlers and Menu Bars" on page 412<br><br>"Hello World Version 4: Adding Dialogs and Push Buttons" on page 420<br><br>"Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help" on page 433<br><br>"Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics" on page 443 | IFrameWindow                 |

| Tasks                                                 | Sample or Example                                                                                                                                                                                                                                                                                                                                                          | Class Usage                                             |
|-------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| Display application status in a status area           | <p>“Hello World Version 3: Add Command Handlers and Menu Bars” on page 412</p> <p>“Hello World Version 4: Adding Dialogs and Push Buttons” on page 420</p> <p>“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433</p> <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p> | <p>IFrameExtension<br/>IFrameWindow<br/>IStaticText</p> |
| Process menu bar items                                | <p>“Hello World Version 3: Add Command Handlers and Menu Bars” on page 412</p> <p>“Hello World Version 4: Adding Dialogs and Push Buttons” on page 420</p> <p>“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433</p> <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p> | <p>ICommandHandler<br/>IMenuBar</p>                     |
| Let the user perform functions using accelerator keys | <p>“Hello World Version 4: Adding Dialogs and Push Buttons” on page 420</p> <p>“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433</p> <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p>                                                                                | <p>IFrameWindow<br/>IMenuBar</p>                        |

| Tasks                                                             | Sample or Example                                                                                                                                                                                                                                                                                                             | Class Usage                                               |
|-------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| Request text information from users using a modal dialog          | <p>“Hello World Version 4: Adding Dialogs and Push Buttons” on page 420</p> <p>“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433</p> <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p>                                   | IFrameWindow                                              |
| Display multiple controls in a client area using a canvas control | <p>“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433</p> <p>“Create a Set Canvas” on page 157</p> <p>“Create a Split Canvas” on page 165</p> <p>“Create a Multicell Canvas” on page 181</p> <p>Set Canvas Sample</p> <p>Split Canvas Sample</p> <p>Multicell Canvas Sample</p> | IMultiCellCanvas<br>ISplitCanvas<br>ISetCanvas<br>ICanvas |
| Display information to a user in a list                           | <p>“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433</p> <p>“Create a List Box” on page 119</p> <p>“Add or Delete a List Box Item” on page 128</p> <p>List Box Sample</p>                                                                                                      | IListBox                                                  |
| Perform an action when a user selects an item from a list         | <p>“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433</p>                                                                                                                                                                                                                       | IListBox<br>IListBox::Cursor<br>ISelectHandler            |
| Display and repaint an IDrawingCanvas                             | <p>“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433</p> <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p>                                                                                                               | IDrawingCanvas<br>IResizeHandler                          |

| Tasks                                                                              | Sample or Example                                                                                                                                                                                                                | Class Usage                                  |
|------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| Add help to an application (requires use of IPFX/X on AIX and Windows)             | <p>“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433</p> <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p>                  | IHelpHandler<br>IHelpWindow                  |
| Perform an action when an user selects an item from a combo box in a dialog window | “Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443                                                                                                                                 | IComboBox<br>ICommandHandler<br>IFrameWindow |
| Show multiple components on a status line                                          | “Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443                                                                                                                                 | IFrameExtension<br>ISplitCanvas              |
| Display a digital clock (long and short form)                                      | <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p> <p>“Extend Event Handling” on page 27</p> <p>Create Your Own Handler Sample</p>                                          | IString<br>ITime<br>ITimer                   |
| Display the date (long and short form)                                             | “Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443                                                                                                                                 | IDate<br>IString                             |
| Let the user dynamically change the font of an application using a font dialog     | <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p> <p>“Create a Toolbar” on page 245</p> <p>“Add a Toolbar” on page 247</p> <p>Toolbar Sample 1</p> <p>Toolbar Sample 2</p> | IFontDialog<br>ISelectHandler                |
| Customize an application using a notebook control                                  | “Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443                                                                                                                                 | ICommandHandler<br>IFrameWindow<br>INotebook |
| Customize an application using a pop-up menu                                       | <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p> <p>“Create a Pop-Up Menu for a List Box” on page 65</p>                                                                  | IMenuHandler<br>IPopUpMenu                   |

| Tasks                                                | Sample or Example                                                                                                                                                                                                                                                                                  | Class Usage                           |
|------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|
| Record a user's settings of an application           | "Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics" on page 443                                                                                                                                                                                                   | IProfile                              |
| Customize a control                                  | "Add Styles" on page 15                                                                                                                                                                                                                                                                            | IBitFlag<br>IWindow                   |
| Create a message box                                 | "Create a Message Box" on page 11<br>Message Box Sample                                                                                                                                                                                                                                            | IMessageBox                           |
| Save data from a user-edited control to a file       | "Create an MLE" on page 92<br>Multi-Line Entry Field Sample                                                                                                                                                                                                                                        | IMultiLineEdit                        |
| Display a list of choices to a user                  | "Create a List Box" on page 119<br><br>"Add or Delete a List Box Item" on page 128<br><br>"Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help" on page 433<br><br>"Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics" on page 443 | IListBox<br>ISelectHandler            |
| Displaying a discrete set of choices to a user       | "Create a List Box" on page 119<br><br>"Add or Delete a List Box Item" on page 128                                                                                                                                                                                                                 | INumericSpinButton<br>ITextSpinButton |
| Arrange child controls in rows or columns            | "Create a Set Canvas" on page 157<br>Set Canvas Sample                                                                                                                                                                                                                                             | ISetCanvas                            |
| Arrange child controls in a grid of rows and columns | "Create a Multicell Canvas" on page 181<br>Multicell Canvas Sample                                                                                                                                                                                                                                 | IMultiCellCanvas                      |
| Provide a scrollable view area                       | "Create a View Port" on page 193<br>View Port Sample                                                                                                                                                                                                                                               | IViewPort                             |

| Tasks                                                                                                                   | Sample or Example                                                                                                                                                                                                                                                                                 | Class Usage                                                                                                                                                           |
|-------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Let the user make a file selection using a file dialog control                                                          | <p>“Specify File Dialog Information” on page 261</p> <p>“Create a File Dialog” on page 263</p> <p>“Specify Font Dialog Information” on page 266</p> <p>“Create a Font Dialog” on page 267</p> <p>Multiline Entry Field Sample</p> <p>Multimedia Remote Sample</p> <p>Multimedia Stereo Sample</p> | <p>IFileDialog</p> <p>IFileDialog::Settings</p>                                                                                                                       |
| Create pop-up menus                                                                                                     | <p>“Create a Pop-Up Menu” on page 62</p> <p>“Create a Pop-Up Menu for a List Box” on page 65</p> <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p> <p>Toolbar Sample 1</p>                                                                  | <p>IMenuHandler</p> <p>IPopUpMenu</p>                                                                                                                                 |
| Create and populate a container control                                                                                 | <p>“Create and Use Containers” on page 225</p> <p>Container Sample</p>                                                                                                                                                                                                                            | <p>ICnrAllocator</p> <p>IContainerControl</p> <p>IContainerObject</p>                                                                                                 |
| Enable direct manipulation support                                                                                      | <p>“Use Default Direct Manipulation” on page 316</p>                                                                                                                                                                                                                                              | <p>IDM* classes</p>                                                                                                                                                   |
| Enable direct manipulation for an entry field or an MLE control                                                         | <p>Direct Manipulation Sample 1</p>                                                                                                                                                                                                                                                               | <p>IDMHandler</p> <p>IEntryField</p> <p>IMultiLineEdit</p>                                                                                                            |
| Enable direct manipulation for intra-process (source and target containers in the same process) container support       | <p>Direct Manipulation Sample 3</p>                                                                                                                                                                                                                                                               | <p>IContainerColumn</p> <p>IContainerControl</p> <p>IContainerObject</p> <p>IDMHandler</p>                                                                            |
| Enable direct manipulation for inter-process (source and target containers in the separate processes) container support | <p>Direct Manipulation Sample 4</p>                                                                                                                                                                                                                                                               | <p>IContainerControl</p> <p>IContainerObject</p> <p>IDMCnrItem</p> <p>IDMHandler</p> <p>IDMSourceDiscardEvent</p> <p>IDMSourceOperation</p> <p>IDMTargetDropEvent</p> |
| Process different events (time changes on a clock)                                                                      | <p>“Extend Event Handling” on page 27</p>                                                                                                                                                                                                                                                         | <p>IHandler</p>                                                                                                                                                       |
| Add mouse handlers                                                                                                      | <p>“Handle Mouse Events” on page 33</p>                                                                                                                                                                                                                                                           | <p>IMouseHandler</p>                                                                                                                                                  |
| Use clipboards in your applications                                                                                     | <p>“Add Clipboard Support” on page 286</p> <p>Clipboard Sample</p>                                                                                                                                                                                                                                | <p>IClipboardHandler</p> <p>IClipboard</p>                                                                                                                            |

| Tasks                                                       | Sample or Example                                                                                                                                                                                                                                                                                                                                                 | Class Usage                                                                                                                                |
|-------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Provide toolbars that the user can manipulate and customize | <p>“Create a Toolbar” on page 245</p> <p>“Add a Toolbar” on page 247</p> <p>Toolbar Sample 1</p> <p>Toolbar Sample 2</p>                                                                                                                                                                                                                                          | <p>IToolBar</p> <p>IFlyOverHelpHandler</p>                                                                                                 |
| Add 2D graphics support to your applications                | <p>2DCombo Graphics Sample</p> <p>2DDocs Graphics Sample</p> <p>2DPaint Graphics Sample</p> <p>BarChart Graphics Sample</p> <p>Magnify Sample</p> <p>“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433</p> <p>“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443</p> | <p>IG* classes</p>                                                                                                                         |
| Create a remote control interface using multimedia classes  | <p>“Create Master Devices” on page 338</p> <p>“Play Audio Compact Discs” on page 340</p> <p>“Create Audio Devices” on page 354</p> <p>“Create Video Devices” on page 378</p> <p>Multimedia Remote Sample</p>                                                                                                                                                      | <p>IMM* classes</p>                                                                                                                        |
| Create a stereo interface using multimedia classes          | <p>“Create Master Devices” on page 338</p> <p>“Play Audio Compact Discs” on page 340</p> <p>“Create Audio Devices” on page 354</p> <p>“Create Video Devices” on page 378</p> <p>Multimedia Stereo Sample</p>                                                                                                                                                      | <p>IMM* classes</p>                                                                                                                        |
| Use a multiline entry field to create a simple editor       | <p>“Create an MLE” on page 92</p> <p>Multiline Entry Field Sample</p>                                                                                                                                                                                                                                                                                             | <p>ICommandHandler</p> <p>IFileDialog</p> <p>IFontDialog</p> <p>IFrameWindow</p> <p>IHandler</p> <p>IMenuHandler</p> <p>IMultiLineEdit</p> |



| Tasks                                                                      | Sample or Example | Class Usage                                               |
|----------------------------------------------------------------------------|-------------------|-----------------------------------------------------------|
| Use a view port canvas displaying a bitmap as a page in a notebook control | Notebook Sample   | IBitmapControl<br>IBitmapHandle<br>INotebook<br>IViewPort |

You can find all the samples in the samples/ioc directory.

**RELATED CONCEPTS**

“Chapter 1. Windows” on page 1

“Chapter 3. Events and Event Handlers” on page 19

**RELATED TASKS**

“Chapter 27. Work with the User Interface Samples” on page 401

“Hello World Version 1: Create a Main Window” on page 402

“Hello World Version 2: Add Resource Files and Frame Extensions” on page 405

“Hello World Version 3: Add Command Handlers and Menu Bars” on page 412

“Hello World Version 4: Adding Dialogs and Push Buttons” on page 420

“Hello World Version 5: Add Canvases, a List Box, Native System Functions, and Help” on page 433

“Hello World Version 6: Adding a Font Dialog, Pop-up Menus, Notebooks, and Graphics” on page 443

“Write an Event Handler” on page 25

“Task and Samples Cross-Reference Table” on page 445