IBM C/C++ Tools: User Interface Class Library Guide Version 2.0

Document Number S71G-2230-00

Note!

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

First Edition (May 1993)

This edition applies to Version 2.0 of IBM C/C++ Tools (Programs 61G1176 and 61G1426) 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.

Requests for publications and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

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

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, Ontario, Canada. M3C 1V7

You can also send your comments by facsimile to (416) 448-6057 addressed to the attention of the RCF Coordinator. If you have access to Internet, you can send your comments electronically to **torrcf@vnet.ibm.com**; IBMLINK, to **toribm(torrcf)**; or IBM/PROFS, to **torolab4(torrcf)**.

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

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

© **Copyright International Business Machines Corporation 1993. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
IBM is a registered trademark of International Business Machines Corporation, Armonk, N.Y.

Notices

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

Trademarks and Service Marks

The following terms used in this publication are trademarks or service marks of IBM Corporation in the United States or other countries. They are denoted by an asterisk (*) when they first appear in the text.

BookManager	OS/2
C/2	Personal System/2
C Set/2	Presentation Manager
Common User Access	PS/2
CUA	SAA
IBM	Systems Application Architecture
Operating System/2	WorkFrame/2

The following terms used in this publication are trademarks or service marks of other corporations in the United States or other countries. They are denoted by a double asterisk (**) when they first appear in the text.

Microsoft	Microsoft Corporation
Pentium	Intel Corporation

© Copyright IBM Corp. 1993

Acknowledgments

This publication is the result of a residency conducted at the International Technical Support Center, Boca Raton, Florida.

The advisor for this project was:

Dieter Neumann International Technical Support Center, Boca Raton, Florida.

The authors of this document are:

Fred Brown IBM Cary

Wayne Chen IBM Taiwan

Roberto Ribeiro dos Santos IBM Brazil

Niroo Thaya-Paran IBM UK

Special thanks to the following people for providing information and technical assistance in the production of this document:

David Lavin C++ User Interface Development, IBM Cary

Stephenie Joyner IBM Cary

Wes Wilson IBM Cary

Kevin Leong IBM San Jose

Maxine Houghton IBM Toronto

Alistair Rennie IBM Toronto

Special thanks to the entire User Interface Class Library team for reviewing and suggesting improvements to this document.

© Copyright IBM Corp. 1993

About This Book

The purpose of the IBM C/C++ Tools: User Interface Class Library (hereafter referred to as User Interface Class Library) is to provide a library of classes that you can use to develop graphical user interfaces for object-oriented applications using the C++ programming language.

The *IBM C/C++ Tools: User Interface Class Library User's Guide* will help you learn some of the basic features the class library provides to help you develop your own applications and will enable you to start using the User Interface Class Library classes. This book assumes that you have C++ programming knowledge and experience. Refer to the *C++ Programmers Guide* to review C++ programming concepts and principles.

The *IBM C/C++ Tools: User Interface Class Library User's Guide* is divided into three parts. Everyone should read Chapter 1, "Introduction" on page 3, which provides an overview of the class libraries and defines some terms used throughout this book. If you are new to C++ class libraries, you'll want to read Part 1 first to gain a general understanding of the benefits that the User Interface Class Library offers. If you have previously used class libraries, you may want to browse Part 1 and go directly to Part 2 to learn about the advanced features you can use in complex applications. To get started using the User Interface Class Library classes, read Part 3, "Sample Applications" on page 137 first. You can use our code samples to learn how to develop your own applications using the User Interface Class Library classes.

Part 1. Learning the Basics

This part introduces some of the key concepts you will want to understand about class libraries.

Chapter 1, "Introduction" on page 3 provides a high-level description of the class library. The classes in the library are grouped into categories based on the tasks you perform when developing applications.

© Copyright IBM Corp. 1993

Chapter 2, “Application Classes” on page 17 describes the classes that make up a typical application and the classes you use to develop basic application components.

Chapter 3, “Window Classes” on page 21 describes the classes that enable you to create frame extensions, basic controls, and canvas controls.

In Chapter 4, “Handlers and Events” on page 61, you will learn about handler and event classes, as well as how to create your own handler class.

Chapter 5, “Data Types and Attributes” on page 69 describes the classes used to manage character data.

Part 2. Beyond the Basics

This part introduces some of the more advanced features of the class library.

Chapter 6, “Advanced Controls” on page 81 describes classes used to create MLEs, containers, and notebook controls.

In Chapter 7, “Advanced Topics” on page 105, you will learn about some of the advanced features of the User Interface Class Library class library (including ways to extend event handling, tips on exception handling, and creating threads) that will enable you to create more complex applications.

Chapter 8, “Finishing Touches” on page 123 discusses the ways to put finishing touches on your applications and describes how to create help and use NLS functions in your applications.

Part 3. Sample Applications

This part of the book contains sample applications that will help you apply what you learned in Parts 1 and 2.

Chapters 9 through 16 take you step-by-step through a simple application that illustrates many of the features of the User Interface

Class Library classes and member functions. You can find sample code for each of the examples on the *User Interface Class Library Samples* directory, so you can follow along and create your own examples as you read this book. You will look at several versions of the "Hello World" application, and each version builds on concepts covered in the previous versions.

Appendix A, "Hierarchy," contains a complete list of the classes, organized by the categories described in the "Introduction."

Appendix B, "Class Library Conventions," lists the guidelines used by the class library to define standard file names, class names, function names, and data member names.

Finally, the Bibliography lists additional sources of information that will help you as you begin developing your own applications with the User Interface Class Library.

While reading this book, you will want to refer to the *IBM C/C++ Tools: User Interface Class Library Reference* (S61G-1179-00) for complete reference details on the classes. That book is available online as part of the product, and you can order a separate hard copy version.

Getting Started

This chapter contains the following information to help you start using the User Interface Class Library:

- Hardware, software, and operating system requirements
- Introduction to the contextual help and online documentation.

Hardware, Software, and Operating System Requirements

The IBM C/C++ Tools product requires a workstation with a 32-bit processor (80386, 80486, or Pentium** microprocessor) running the OS/2* 2.0 or later operating system.

The OS/2 2.0 Developer's Toolkit, referred to in this document as the Toolkit, is also a prerequisite, primarily because it contains the system linker that the compiler uses, as well as the system header files and import libraries that increase the capabilities of the compiler, and the IMPLIB utility that helps manage the build process for projects.

The IBM C/C++ Tools Version J2.0 product requires a workstation running the IBM OS/2 Version J2.0 operating system. The IBM OS/2 Developer's Toolkit Version J2.0 is also a prerequisite.

To effectively use the C/C++ Tools compiler and debugger, you need a minimum of 8M of RAM for C applications and 12M for C++ applications. You must also set your swap path to a directory with at least 10MB free for C applications or 14M for C++ applications. A full installation of the C/C++ Tools or C/C++ Tools Version J2.0 files requires about 30MB of disk space, broken down in the following manner:

```
Compiler and libraries       8.  MB
Debugger                   6.  MB
EXTRA                     2.  MB
Browser                   2.  MB
Standard class libraries      .5 MB
Collection class library     1.  MB
User Interface class library  5.  MB
Online information          4.5 MB
WorkFrame/2* support         1.  MB
```

© Copyright IBM Corp. 1993

When you install the product, the installation program tells you how much space you have available on the selected drive and how much space is required for the options you select.

If you have an 80386 processor, an 80387 math coprocessor is recommended because it will greatly increase the speed of floating point operations. If you have an 80486SX processor, an 80487 math coprocessor is recommended.

About the Contextual Help Feature

The User Interface Class Library provides contextual help for each class and member function. The requirements for accessing contextual help are:

- The DDE4UIL.INF and DDE4UIL.NDX files must be installed and available.
- You must use the Enhanced System Editor provided by OS/2 Version 2.0.

If these requirements are met, you can access contextual help from the Enhanced System Editor by positioning the cursor over the name of a class or member function in the text being edited and pressing the Ctrl-H keys. This opens the online version of the *IBM C/C++ Tools: User Interface Class Library Reference* and displays information about the class or member function under the cursor.

For complete details on setting the environment variables needed to use the contextual help feature, refer to the installation instructions for this product.

About the Samples

Many of the samples described in this document are shipped with the User Interface Class Library product. To find these samples, look in the `\ibmcpp\samples\iclui` directory.

Contents

Notices . . . iii
Trademarks and Service Marks . . . iii

Acknowledgments . . . v

About This Book . . . vii
Part 1. Learning the Basics . . . vii
Part 2. Beyond the Basics . . . viii
Part 3. Sample Applications . . . viii

Getting Started . . . xi
Hardware, Software, and Operating System Requirements . . . xi
About the Contextual Help Feature . . . xii
About the Samples . . . xii

Part 1. Learning the Basics . . . 1

Chapter 1. Introduction . . . 3
Overview of the Class Library . . . 3
Overview of the Classes . . . 4
A Simple Class Library Application . . . 10
A Sample C++ Source File . . . 12
A Sample Resource File . . . 15
Using What You've Learned . . . 16

Chapter 2. Application Classes . . . 17
Command Line Parameters . . . 17
Run and Exit . . . 18
String Resources . . . 18
User Resource Files . . . 19

Chapter 3. Window Classes . . . 21
Frame Extensions and Resources . . . 21
Title Bar . . . 23
The Minimized Icon . . . 24
Menu Bar . . . 24

© Copyright IBM Corp. 1993

Information Area 28
Status Area 30
Basic Controls 31
Static Text Control 31
Entry Field Control 33
Push Button Control 34
Check Box Control 36
Radio Button Control 38
Slider Control 40
Canvas Controls 44
Split Canvas 45
Set Canvas 47
Multicell Canvas 51
Viewport 54
Styles 55
Style Objects 56
Setting Window Styles 56
Cursors 58

Chapter 4. Handlers and Events 61
Handlers and Events 61
Handlers 62
Events 65
Writing a Handler 67

Chapter 5. Data Types and Attributes 69
Managing Character Data 69
Stream I/O 69
Accessors 70
Testing 71
Comparison 72
Conversion 74
Modifying and Aligning 75
Manipulation 76
Fonts 77

Part 2. Beyond the Basics 79

Chapter 6. Advanced Controls 81

Multiple-Line Entry Field Control 81
Creating an IMultiLineEdit Instance 81
Loading and Saving a File 83
Positioning the Cursor 83
Clipboard Operations 85
Container Control 86
Creating a Container 87
Creating an Instance of a Container Object 87
Adding and Removing Objects 89
Filtering Objects 90
Cursors and Containers 93
Working with Views 95
Container Columns and Details View 97
Creating a Pop-up Menu in a Container 100
Notebook Control 101
Notebook Styles 102
Page Settings 104

Chapter 7. Advanced Topics 105
Extending the Event Handling 105
Tracing 109
Exception Handling 112
Providing a Default Exception Handler 114
Threads and Protecting Data 116
Current Thread 116
Starting a Thread 117
Protecting Data 121
Critical Sections 122

Chapter 8. Finishing Touches 123
Standard Dialogs 123
File Dialog 123
Font Dialog 125
Message Box 127
Pop-Up Menus 128
Using Help 130
DBCS and NLS Support 133

Part 3. Sample Applications 137

Chapter 9. Introduction to the Sample Applications 139
Running the Samples 139
Conventions Overview 140
Other Conventions Used in the Sample Code 140

Chapter 10. Class Library Applications 141
Structure of User Interface Class Library Applications 141
Creating Your Own Classes 143

Chapter 11. A Simple Application with a Main Window . . . 145
Version 1 Window Parent Relationship Diagram 146
Version 1 Files 147
The Source Code File 147
The Module Definition File 148
Tasks Performed by Version 1 148
Creating the Main Window 148
Creating a Static Text Control for Version 1 149
Putting Text in the Static Text Control 150
Aligning Text within the Static Text Control 150
Setting the Control as the Client Window 150
Setting the Focus and Showing the Main Window 151
Running the Application 151
Compiling and Linking Version 1 152
The Module Definition File Format 153

Chapter 12. Adding a Resource File and Frame Extensions 155
Version 2 Window Parent Relationship Diagram 156
Version 2 Files 158
The Primary Source Code File 158
The AHelloWindow Class Header File 160
The Constants Definition File 161
The Resource File 162
The Icon File 162
AHELLOW2.DEF 163
Advantages of the C++ File Structure 163
Tasks Performed by Version 2 164
Creating the Main Window 164
Getting the Current Application and Running It 166
Constructing the Main Window 166

Chapter 13. Event Handling and Menu Bars 173
Version 3 Window-Parent Relationship Diagram 174
Version 3 Files 175
The Primary Source Code File 176
The AHelloWindow Class Header File 178
The Constants Definition File 179
The Resource File 180
The Icon File 181
AHELLOW3.DEF 182
Tasks Performed by Version 3 182
Creating a Status Area Using a Static Text Control 183
Putting Text in a Static Text Control for a Status Line 183
Specifying the Location and Height of the Status Area 184
Setting AHelloWindow as the Event Handler 184
Creating a Menu Bar 185
Setting an Initial Check Mark in the Pull-down Menu 186
Adding Command Processing to Align the Static Text 186
Compiling and Linking Version 3 187

Chapter 14. Simple Dialogs and Push Buttons 189
Version 4 Window-Parent Relationship Diagram 190
Version 4 Files 191
The Primary Source Code File 192
The AHelloWindow Class Header File 197
The Constants Definition File 198
The Text Dialog Source Code File 199
The ATextDialog Class Header File 201
The Resource File 202
The Icon File 204
The Text Dialog Template 204
The Text Dialog Resource File 204
AHELLOW4.DEF 205
Tasks Performed by Version 4 205
Modifying the Menu Bar and Pull-down Menu 206
Adding Push Buttons in a Set Canvas 211
Compiling and Linking Version 4 215

Chapter 15. Canvas, User-Created Control, and Help 217
Version 5 Window-Parent Relationship Diagram 218
Version 5 Files 219

The Primary Source Code File . . . 219
The AHelloWindow Class Header File . . . 225
The Constants Definition File . . . 226
The Text Dialog Source Code File . . . 227
The ATextDialog Class Header File . . . 228
The Earth Window Source File . . . 229
The AEarthWindow Class Header File . . . 230
The Resource File . . . 230
The Icon File . . . 232
The Text Dialog Template . . . 233
The Text Dialog Resource File . . . 233
The Help Window Source File . . . 233
The Module Definition File . . . 239
Tasks Performed by Version 5 . . . 240
Constructing the Main Window Using Newly Defined Member Functions . . . 241
Compiling and Linking Version 5 . . . 249

Chapter 16. NLS and Advanced Functions . . . 251
Version 6 Window-Parent Relationship Diagram . . . 253
Version 6 Files . . . 254
Tasks Performed by Version 6 . . . 255
Compiling and Linking Version 6 . . . 256

Appendix A. Hierarchy . . . 259
Application Classes . . . 259
Window Classes . . . 260
Handler Classes . . . 261
Event Classes . . . 262
Event Classes - DDE Events . . . 263
Data Types and Attributes Classes . . . 264
Settings and Styles Classes . . . 265
Support Classes . . . 266
Exception and Error Handling Classes . . . 267

Appendix B. Class Library Conventions . . . 269
File Names . . . 269
Class Names, Function Names, and Data Member Names . . . 270
Enumerations . . . 271
Function Return Types . . . 271

Function Arguments 272
Other Standards 272

Bibliography 273
The IBM C/C++ Tools Library 273
C and C++ Related Publications 273
IBM WorkFrame/2 Publications 273
IBM OS/2 2.0 Publications 273
IBM OS/2 2.0 Technical Library 274
Other Books You Might Need 274
BookManager* READ/2 Publications 274
Systems Application Architecture* Publications 274

Glossary 275

Index 283

Part 1. Learning the Basics

© Copyright IBM Corp. 1993

Chapter 1. Introduction

The User Interface Class Library is an Object Oriented (OO) C++ class library that simplifies the construction of OS/2 applications with graphical user interfaces (GUI). You can use the library to build efficient applications that support Common User Access* (CUA). Workplace look and feel, and that take advantage of all the features of OS/2 Presentation Manager* (PM).

Overview of the Class Library

Here are some of the key features of the class library:

- A rich set of window and control classes that use the base operating system controls. Applications created with this class library have the same appearance and behavior as other OS/2 Presentation Manager Applications.
- Additional functions above the base controls, including the canvas classes and the information area control. The canvas classes help developers with window layout and eliminate the need to specify absolute screen positions and sizes.
- A set of event and handler classes. These classes give developers added control and flexibility in event handling.
- Support for easily adding window frame extensions, such as a status line, to a frame window.
- A set of supporting classes to help with the total application. These include classes to help manage resources and threads.
- Support for writing National Language Support (NLS) and Double Byte Character Support (DBCS) enabled applications. The string and canvas classes assist developers with these tasks.
- A rich set of window styles to change the appearance or behavior of the windows. In addition, defaults are provided to make writing applications easier.
- Support for C++ exception handling. Classes are also provided to help with tracing your application.

© Copyright IBM Corp. 1993

Overview of the Classes

The class library contains over 260 classes and over 2600 member functions. To assist you in learning about the classes and to guide you as you start developing applications, we've grouped the classes into eight basic categories:

1. Application
2. Window
3. Handler
4. Event
5. Data Types and Attributes
6. Settings and Styles
7. Supporting classes
8. Exception and Error Handling

While most applications will use classes from all of these categories, the focus of this book will be on the window category and its relationship with the event, handler, and the other categories.

The application-related classes provide support for the application, threads, profiles and the resources used by the application.

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

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

- Handler classes allow the developer to change the behavior of a control or window class without subclassing this control.
- Handlers can be dynamically added and deleted to a window. The developer can dynamically change the behavior a window by dynamically adding and removing handlers assigned to a window.
- The developer can provide common behavior and code reuse by using a single handler for more than one window.
- Multiple handlers can be attached to a single window.

The data types and attributes classes model basic data types such as strings, points and rectangles. These classes hide the structure of the data, while providing the capability to access and alter the data. Attributes classes are used to specify the color or font to be used within a given window. In addition, a set of handle classes are provided to access window or application specific handles.

Settings and style classes allow the application developer to change the appearance or behavior of window classes. For example, the `IFontDialog` class allows the developer to specify the default font and window title to be displayed. Setting the text alignment, specifying a minimize button on a frame, or specifying a tab stop are some examples of styles that can be set by the developer.

Supporting classes are used in a collaborative manner with other classes. Examples of these classes are cursors and program-to-program communication using dynamic data exchange (DDE).

The library creates exception objects to inform the application that the library cannot complete a request. Instances of these classes capture the type of exception and other information about the exception.

The next two pages provide a summary of the classes in each category.

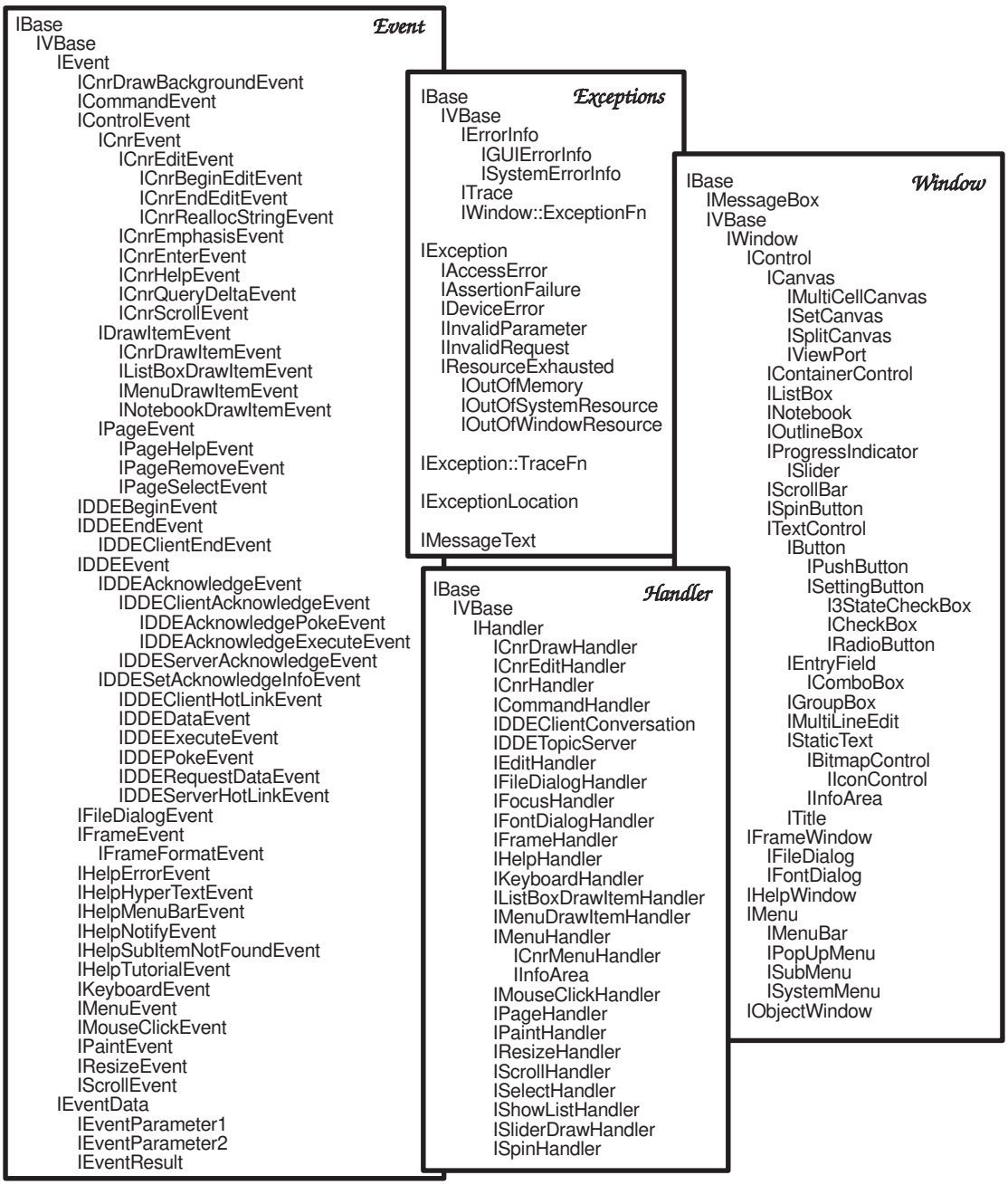


Figure 1. Event, Handler, Exception and Window Classes

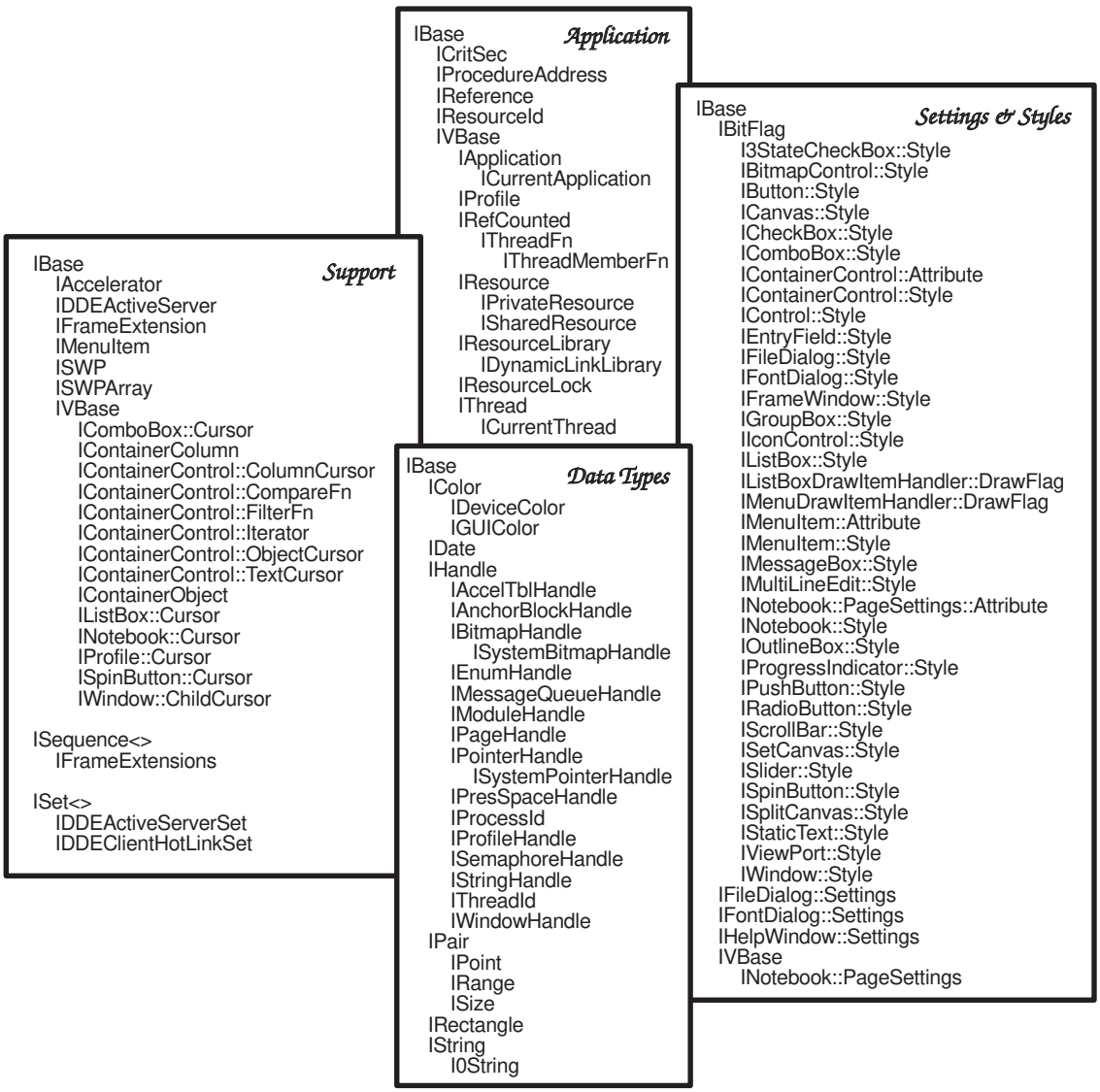


Figure 2. Data Types, Support, Application and Settings & Styles

To learn more about a specific category, refer to the sections listed with each category:

Categories	References
Application	Chapter 2, “Application Classes” on page 17
	“Threads and Protecting Data” on page 116
	“Application Classes” on page 259
Window	Chapter 3, “Window Classes” on page 21
	Chapter 6, “Advanced Controls” on page 81
	Chapter 8, “Finishing Touches” on page 123
	“Window Classes” on page 260
Handler	Chapter 4, “Handlers and Events” on page 61
	“Handler Classes” on page 261
Event	Chapter 4, “Handlers and Events” on page 61
	“Extending the Event Handling” on page 105
	“Event Classes” on page 262
Data types and attributes	Chapter 5, “Data Types and Attributes” on page 69
	“Data Types and Attributes Classes” on page 264
Settings and styles	“Styles” on page 55
	“Settings and Styles Classes” on page 265

Supporting classes

"Cursors" on page 58

"Support Classes" on page 266

Exception and error handling

"Exception Handling" on page 112

"Tracing" on page 109

"Exception and Error Handling Classes" on page 267

A Simple Class Library Application

An easy way to understand how the different classes and objects you've just read about work together is to look at a simple application. The application has three basic user interface components:

- A standard frame window with a title bar, minimized icon, borders, minimize and maximize buttons. The window title will be set to "Simple Application".
- A Menu Bar that contains a single menu item called "Close". When the user selects this menu item, the application will close the window and terminate.
- The rest of the window or client area that contains the phrase "Simple Example".

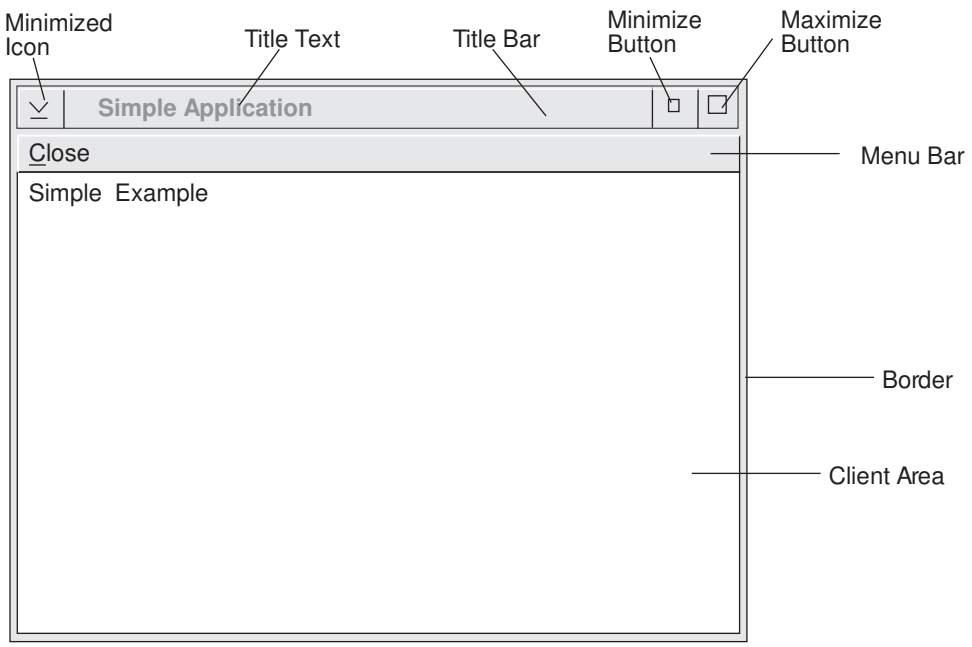


Figure 3. Simple Application Main Window

Two source files are required to write this application. The first file is the C++ source file used by the C++ compiler to generate the executable part of this application. The second file contains the

resource definitions used by the resource compiler to generate the resources for this application. It is not important that you understand every line in these files or the steps to create an application. Let's focus on learning the key concepts.

A Sample C++ Source File

Listing of the C++ source file:

```
 1 #include <iapp.hpp>                 //IApplication Class
 2 #include <iframe.hpp>               //IFrameWindow Class
 3 #include <icmdhdr.hpp>              //ICommandHandler & ICommandEvent
 4 #include <istattxt.hpp>             //IStaticText Class
 5 #include <istring.hpp>              //IString Class
 6
 7 #define WND_MAIN        5           //Main Window Id
 8 #define MI_CLOSE        5  1        //Push Button Window Id and Command Id
 9
1  class AWindow : public IFrameWindow,    //Define AWindow Class from
11              public ICommandHandler  //  IFrameWindow & ICommandHandler
12 {
13 public:
14   AWindow(unsigned long windowId)      //Define AWindow Constructor
15     : IFrameWindow (                  //Call IFrameWindow constructor
16       IFrameWindow::defaultStyle()     //  Use default styles plus
17       | IFrameWindow::menuBar,         //  Get Menu Bar from Resource File
18       windowId)                      //  Main Window Id
19   {
2      IString aString("Simple Example");  //Create text string for static text
21     IStaticText staticText(MI_CLOSE,    //Create Static Text Control
22       this, this);                    //  Pass in myself as parent & owner
23     staticText.setText(aString);       //Set text in Static Text Control
24     handleEventsFor(this);             //Set self as command event handler
25     setClient(&staticText);            //Set button control in Client Area
26     setFocus();                        //Set focus to main window
27     show();                           //Set to show main window
28   } /* end AWindow :: AWindow(...) */
29
3    Boolean command(ICommandEvent& cEvent)//Define command member function
31   {
32     if (cEvent.commandId() == MI_CLOSE) //Is Command Event Id = Close Id
33     {                                 //  Yes, the command is close
34       close();                         //  Let's close the main window
35       return true;                     //  Normally, you would return true
36     };                                //  to indicate command processed
37     return false;                      //Return Command not Processed
38   } /* end AWindow :: command(...) */
39 };                                   // End of AWindow class definition
4
41 void main()                           //Main Procedure with no parameters
42 {
43   AWindow mainWindow(WND_MAIN);        //Create main window on the desktop
44   IApplication::current().run();       //Get current application & start
45 } /* end main */
```

Lines 1-5 include the class header files needed from the class library for the application. WND_MAIN (line 7) is used as the window ID for the main window. MI_CLOSE (line 8) is used as the command ID for the "Close" menu item.

A class called `AWindow` is defined in lines 10-39. This class is derived from the `IFrameWindow` and `ICommandHandler` classes (lines 10-11). The `AWindow` class has a single constructor (lines 14-28) and a single member function called `command` (lines 30-39). Many objects are created when this application is run; however, the application does not need to know about most of these objects. A list of the key objects created by running the application follows:

Object Name	Details about the Object
mainWindow	This AWindow object is the main window for the application. It is created on line 43.
staticText	This is the static text control (IStaticText) object that will contain the phrase "Simple Example". This object is created on line 21.
aString	This IString object is created on line 20. It contains the phrase "Simple Example" that will be set in the staticText object on line 23.
title bar	This object is created by the class library because we specified the default styles on the IFrameWindow contrustor on lines 15-16.
menu bar	This object is created by the class library as a result of specifying the *menuBar* style on the IFrameWindow construtor on lines 15-17. Several handlers are also created supporting the menu bar. In this application, the default command handler for this menu bar will send the command events to the frame window. Line 24 specifies that we are a command handler for ourselves. Lines 30-38 define the processing for these command events.
cEvent	This ICommandEvent object is created by the class library and is a parameter on the command member function on line 30. This object returns the command ID on line 32 and is compared against MI_CLOSE. If it

is an MI_CLOSE command, the application closes the window and terminates using the *close* on line 34; otherwise, the member function returns false indicating that the command has not been processed.

A Sample Resource File

Now, let's list the resource file for this simple application:

```
 1 #define WND_MAIN        5             //Main window Id
 2 #define MI_CLOSE        5  1          //Push button window Id & Command Id
 3
 4 STRINGTABLE
 5   BEGIN
 6     WND_MAIN,   "Simple Application"    //Title bar text (main Id)
 7   END
 8
 9 MENU WND_MAIN                         //Main window menu (WND_MAIN)
1    BEGIN
11     MENUITEM "~Close",   MI_CLOSE      //Close menu item
12   END
```

Line 1 defines WND_MAIN for this resource file. This number (5000) needs to match the definition on line 7 of the C++ source file. Line 2 defines MI_CLOSE for this resource file. This number (5001) needs to match the definition on line 8 of the C++ source file.

Line 6 defines a string resource containing the phrase "Simple Example" with a string ID of WND_MAIN. Because this matches the main window ID used on line 43 of the C++ source program, the class library uses this string as the default window title. Note, the "Simple Example" phrase can changed without changing the application code.

Lines 9-12 define the menu bar used by this application. Since we specified the menuBar style on line 17, the class library attempts to load the menu with an ID equal to the window ID. In this example, the main window ID is WND_MAIN (5000) and line 9 defines the menu with a menu ID of WND_MAIN; therefore, the class library uses this menu bar for the main window.

The close menu item is defined on line 11. This menu item appears to the user as **Close** with a command ID of MI_CLOSE. When the user selects this menu item, lines 30-38 in the C++ source code are executed. An important point in this example is that the "Close" phrase could be changed (e.g., "Quit") and the application code would not be required to change. This is important if your application will be translated into other languages or for changes required by end users. It is also possible to reorganize complex menus with many menu items and submenus without changing the application code.

Using What You've Learned

This chapter described the benefits of the Class Library to C++ programmers developing OS/2 applications with graphical user interfaces (GUIs). It defined the general categories of classes that are included in the User Interface Class Library. In addition, it illustrated some concepts and principles using the basic classes and member functions in a simple application. You are now ready to learn more about the classes in the class library and how you can use them to make your application programming tasks easier and more effective.

Chapter 2. Application Classes

To develop an User Interface Class Library application, you always use at least two classes: `IApplication` and `ICurrentApplication`.

Use the `IApplication` class member functions to :

- Assign a name to a process
- Query the name of a process
- Set the priority of a process
- Query the priority of a process.

Use the `ICurrentApplication` class member functions to:

- Save the command line parameters to the program
- Start event processing
- Exit from an application
- Assign resources to an application
- Query resources from an application.

Command Line Parameters

`ICurrentApplication` provides functions to record and query the command line arguments of your application. The arguments are set by calling `setArgs` and passing in the arguments that were passed to main. An example of saving the command line parameter follows:

```
void main(int argc, char **argv)       //Main procedure with parameters
{
  IApplication::current().setArgs(argc, argv);
  ... rest of program
```

where *argc* is the number of parameters received and *argv* is the actual parameters.

Use the member function `argc` to query the number of parameters. This member function always returns a non-zero value because it always has at least the name of the program as a parameter.

To get the *n*th parameter, use the member function `argv`, where the `argv( )` component is always the name of the program. Because `argv`

© Copyright IBM Corp. 1993

is returned as an `IString`, you can use all the overloaded operators for this class. See “Managing Character Data” on page 69 and refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for more information about the `IString` class.

Run and Exit

To start the event processing of any C++ program using the User Interface Class Library you can use the `run` member function of the `ICurrentApplication` class. An example of the statement is highlighted in the following program:

```
void main()                          //Main procedure with no parameters
{
  AHelloWindow mainWindow (WND_MAIN);   //Create our main window on the desktop
  IApplication::current().run();          //Get current & run the application
} /* end main */
```

To exit from the program, use the member function `exit`, as shown below:

```
if (IApplication::current().argv(1)=="") { //If no command line parameters
    IApplication::current().exit();          //Get current & exit the application
  } /* endif */
} /* end AHelloWindow :: AHelloWindow(...) */
```

String Resources

The OS/2 Presentation Manager allows you to collect certain application resources in a resource file. The resource file is compiled and stored either in the application’s executable file or in a dynamic-link library (DLL). You can replace these application resources by simply changing and recompiling the resource file. The benefits of separating your application components are obvious. You can alter items such as text on the title bar or in a menu without changing the main application code.

You can enable your applications for National Language Support by storing the resources for each language in a separate resource file. You can then build your applications as separate executable versions for each language or as a separate DLL for each language.

The User Interface Class Library provides efficient methods for maintaining application resources, such as bit maps, menus, and title text. Each static string used in a window has a corresponding string ID that is defined in a resource file. To alter these text strings, you must change only the strings defined in the resource file. The source programs do not have to be updated. That is, by using the User Interface Class Library and string resource architecture, you can port OS/2 Presentation Manager applications to different national languages by translating only the resource file.

The menu, string table, and dialog template are all defined in the resource file by resource script format.

You can change the title text from the resource file directly and use the resource compiler, RC.EXE, provided with the IBM C/C++ Tools: Compiler, to compile the resource file and link the resource, *.RES, to your executable file.

User Resource Files

Using the User Interface Class Library, you can load a resource from a DLL. Use the `ICurrentApplication` member function `setUserResourceLibrary` to define the specific resource to be loaded in the program. An example is highlighted in the following program:

```
void main(int argc, char **argv)        //Main procedure with no parameters
 {
  IApplication::current().            //Get current
    setArgs(argc, argv);             //  and set command line parameters

  IString Dllname(IApplication::current().argv(1));

  IApplication::current().                  //Get current application
   setUserResourceLibrary(Dllname.asString()); //   Set the name of resource DLL       .

  AHelloWindow mainWindow (WND_MAIN); //Create our main window on the desktop

  IApplication::current().run();       //Get current & run the application

} /* end main */
```

In addition, you can use a different resource DLL to support multiple languages without changing the program. Once you change the DLLs, all resource strings are changed for the language chosen. See

Chapter 16, “NLS and Advanced Functions” on page 251 for an example.

You can also query which user resource library is active at a specific moment. To query for the active user resource library specify the member function `userResourceLibrary`:

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow(windowId)          //Call IFrameWindow constructor
{
  hello = new IStaticText(WND_HELLO,   //Create static text control
    this, this);                    //  Pass in myself as owner & parent
  hello->setText(
   IApplication::current().
   userResourceLibrary().asString());

                                  //Set text in static text control
  hello->setAlignment(              //Set alignment to center in both
   IStaticText::centerCenter);        //  directions
  setClient(hello);                 //Set hello control as client window

  setFocus();                      //Set focus to main window
  show();                         //Set to show main window
} /* end AHelloWindow :: AHelloWindow(...) */
```

Chapter 3. Window Classes

Frame Extensions and Resources

In addition to the standard Presentation Manager frame controls, the User Interface Class Library allows you to add *extensions* to the standard frame window. These extensions are additional controls placed in specific locations (relative to the title bar, menu bar, or client area) and are primarily application specific.

Use the `IFrameWindow` class to create OS/2 *frame* windows. Frame windows are usually children of the desktop and parent/owner windows of the controls in the User Interface Class Library A member function in the `IFrameWindow` class, called `addExtension`, enables you to add a control as a frame extension. The arguments of `addExtension` indicate where the extension is to be located and what portion of the location control is to be allocated to this extension. The *information area*, implemented by the `IInfoArea` class, is an example of such an extension. See “Information Area” on page 28 and refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for more information about information areas.

Conceptually, the `IFrameWindow` class is composed of a number of child windows, including a system menu symbol, a title bar, minimize and maximize buttons, a border, a menu bar, and a client area. The frame's client area is the rectangular portion of the frame window not occupied by the other frame controls. The client window is the window occupying the client area. Figure 4 on page 22 shows the components of a frame window created using the `IFrameWindow` class.

© Copyright IBM Corp. 1993

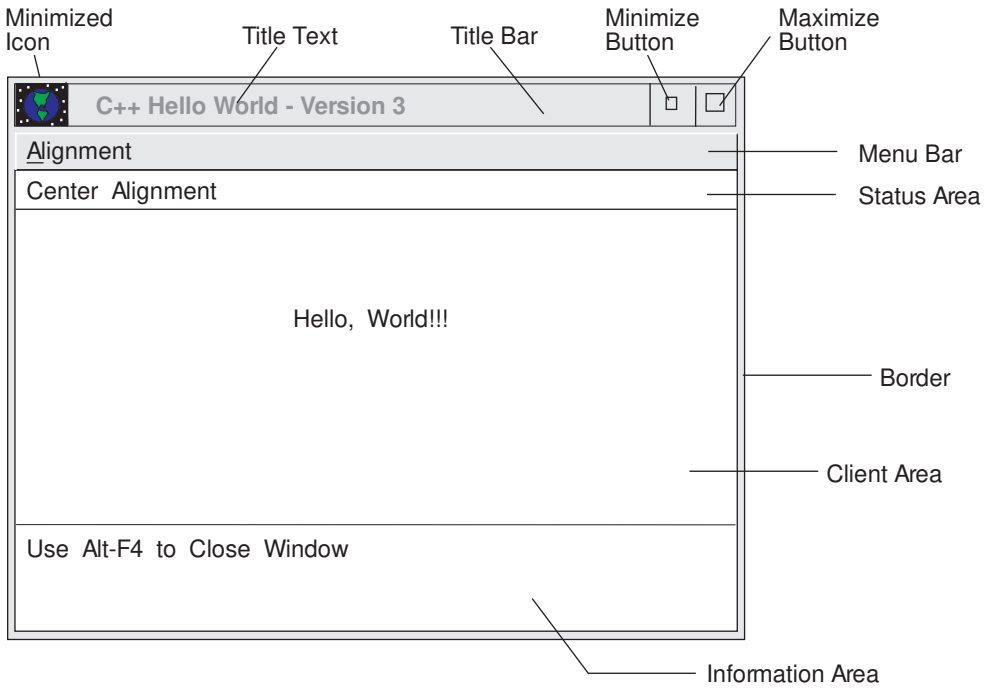


Figure 4. Frame window components

The default style of the `IFrameWindow` class has a title bar, a system menu symbol, a minimize button, a maximize button, and a sizing border. The style also specifies that an entry for this frame will be added to the system window list after this object is created. The `IFrameWindow` also provides several styles, including :

accelerator The frame has an associated accelerator key table

minimizedIcon The frame has an associated icon to be used when minimized

maximized The frame window is created in the maximized state

minimized The frame window is created in the minimized state.

For more information about the styles of `IFrameWindow`, refer to the *IBM C/C++ Tools: User Interface Class Library Reference*.

Title Bar

The title bar is the area at the top of each window that contains the system menu symbol, a minimized icon, a window title, and the maximize and minimize buttons. If you create a minimized icon for your application, you can us it to replace the system menu symbol in your application windows. When a window is maximized, selecting the restore button restores the window to its previous size. The color of the window title bar changes when it receives the input focus.

In the `IFrameWindow` class, you can optionally specify the frame window title. If you do not explicitly provide a title, your application attempts to set the title to a string loaded from the application's resource library using the frame ID. If a string cannot be found, the title defaults to the system-generated title (typically, the name of the executable).

The following example shows how to create the title text by creating the `AHelloWindow` class as a subclass of the `IFrameWindow` class:.

```
#define WND_MAIN   x1
void main()
{
  AHelloWindow mainWindow(WND_MAIN);
  IApplication::current.run();
}
AHelloWindow :: AHelloWindow(unsigned long windowId)
  :IFrameWindow(windowId)
{
  setFocus();
  show();
}
```

And the title text appears in the resource file as:

```
STRINGTABLE
  BEGIN
    WND_MAIN,   "Title Text Sample"
  END
```

The result is a default style frame window with "Title Text Sample" as the title text. WND_MAIN is the frame window identifier. The application passes this identifier to the frame window and uses it to load icon, title, menu, or accelerator table resources from the

application's resource file (if these components are specified using the frame style). In the example, the application passes the title text into the `IFrameWindow` using the WND_MAIN frame ID.

The Minimized Icon

When you create a minimized icon for your application, the minimized icon is placed on the leftmost position in the title bar to replace the system menu symbol.

To add a minimized icon to the title bar, add the icon information to the resource file as follows:

```
ICON WND_MAIN HELLO.ICO
STRINGTABLE
  BEGIN
    WND_MAIN,   "Title Bar with Minimized Icon Sample"
  END
```

Next, modify the `IFrameWindow` constructor to add the `minimizedIcon` style into the default style, as shown below:

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow (
    IFrameWindow::defaultStyle()
    | IFrameWindow::minimizedIcon,
     windowId)
{
  setFocus();
  show();

}
```

When the `AHellowWindow` object is created, its `IFrameWindow` base class is constructed using the default style with a minimized icon, named HELLO.ICO, and "Title Bar with Minimized Icon Sample" as the title text. See "Styles" on page 55 for more information on setting styles.

Menu Bar

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 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.

To add a menu bar to a window, define the contents of the menu bar in the resource file and add the “menubar” into the default style frame window.

In the resource file, define a menu bar with only one submenu, named Alignment:

```
MENU WND_MAIN
  BEGIN
    SUBMENU "~Alignment", MI_ALIGNMENT
      BEGIN
        MENUITEM "~Left",   MI_LEFT
        MENUITEM "~Center", MI_CENTER
        MENUITEM "~Right",  MI_RIGHT
      END
  END
```

When you run the sample and select the **Alignment** choice, an associated pull-down menu is displayed. There are three choices in the pull-down menu: **Left**, **Center**, and **Right**. When one of the choices is selected, the text string in the client window aligns to the position that was selected.

First, define the `IMenuBar` object to the `AHelloWindow` class:

```
class AHelloWindow : public IFrameWindow,
                     public ICommandHandler
{
  public:
    AHelloWindow(unsigned long windowId);
    virtual ~AHelloWindow();
  protected:
    Boolean command(ICommandEvent& cmdEvent);
  private:
    IStaticText    hello;            //Hello contains "Hello, World" text
    IMenuBar       menuBar;
};
```

When the `AHelloWindow` object is created, the `IFrameWindow` and `menuBar` are constructed sequentially.

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow (
    IFrameWindow::defaultStyle()
    | IFrameWindow::minimizedIcon,
    windowId),
    menuBar(WND_MAIN,this),
    hello(WND_HELLO,this,this)
{
  hello.setText(STR_HELLO);
  hello.setAlignment(IStaticText::centerCenter);
  setClient(&hello);
  handleEventsFor(this);
  menuBar.checkItem(MI_CENTER);

  ......
}
```

In the preceding example, a static string is shown in the center of the client area. The statement `handleEventsFor(this)` is used to handle upcoming events. The statement `menuBar.checkItem(MI_CENTER);` places a check mark beside the **Center** menu item.

A command handler is added to handle the selection of menu items.

```
Boolean AHelloWindow :: command(ICommandEvent & cmdEvent)
{
  switch (cmdEvent.commandId()) {
    case MI_CENTER:
      hello.setAlignment( IStaticText::centerCenter);
      menuBar.checkItem(MI_CENTER);
      menuBar.uncheckItem(MI_LEFT);
      menuBar.uncheckItem(MI_RIGHT);
      return(true);
      break;

    case MI_LEFT:
      hello.setAlignment( IStaticText::centerLeft);
      menuBar.uncheckItem(MI_CENTER);
      menuBar.checkItem(MI_LEFT);
      menuBar.uncheckItem(MI_RIGHT);
      return(true);
      break;

    case MI_RIGHT:
      hello.setAlignment( IStaticText::centerRight);
      menuBar.uncheckItem(MI_CENTER);
      menuBar.uncheckItem(MI_LEFT);
      menuBar.checkItem(MI_RIGHT);
      return(true);
      break;
}
```

The alignment of the static text in the client area is adjusted according to the selected menu item, and a check mark is placed in the front of the selected menu item.

See Chapter 4, “Handlers and Events” on page 61 for more information about event handling.

Information Area

The information area is a small rectangular area that is usually located at the bottom of a window. The information area is used to:

- Display a brief explanation of the state of an object
- Display brief help information
- Display information about the completion of a process.

Use the `IInfoArea` class to create and manage the information area. The objects of this class provide a frame extension to show information about the menu item at which the selection cursor is currently positioned. The string displayed in the information area is defined in a resource string table. The string resource is obtained by using the same identifier as the menu item plus some optional offset value (the default offset is 0) that is added to the menu item ID to locate the corresponding string of information about it.

To create an instance of this class, you can use one of the following constructors:

```
IInfoArea ( IFrameWindow *frame, unsigned long id =   );
IInfoArea ( IFrameWindow *frame, unsigned long id, const char *resDLLName );
IInfoArea ( IFrameWindow *frame, const IModuleHandle &resMod,
            unsigned long id =   );
IInfoArea ( IFrameWindow *frame, const char *resDLLName,
            unsigned long id =   );
```

where `IFrameWindow *frame` points to the frame window to which the information area will be attached. The `unsigned long id` is an ID of the information area control (the default value is 0). The `const char *resDLLName` specifies the resource library from which the information strings are to be loaded. The default is the user resource library in the current application class.

The following example uses the IInfoArea class to create the information area. The menu bar and string table in the resource file are defined as follows:

```
MENU WND_MENU
  BEGIN
    SUBMENU "~Alignment", MI_ALIGNMENT
      BEGIN
        MENUITEM "~Left",   MI_LEFT
        MENUITEM "~Center", MI_CENTER
        MENUITEM "~Right",  MI_RIGHT
      END
  END
STRINGTABLE
  BEGIN
    MI_ALIGNMENT "Select Alignment Menu"
    MI_LEFT      "Select Left Alignment Menu Item"
    MI_CENTER    "Select Center Alignment Menu Item"
    MI_RIGHT     "Select Right Alignment Menu Item"
  END
```

Define an object infoArea in the AHelloWindow class, for example:

```
class AHelloWindow : public IFrameWindow,
                     public ICommandHandler
{
  public:
    AHelloWindow(unsigned long windowId);
  protected:
    Boolean command(ICommandEvent& cmdEvent);
  private:
    IMenuBar     menuBar;
    IStaticText  hello;
    IInfoArea    infoArea;
};
```

Modify the constructor of AHelloWindow. The information area is constructed when AHelloWindow is created.

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow ( IFrameWindow::defaultStyle()
                 | IFrameWindow::minimizedIcon, windowId),
    menuBar(WND_MENU,this),
    hello(WND_HELLO,this,this),
    infoArea(this)
{
      ...
}
```

When you choose the menu item, the string related to the chosen menu item is displayed automatically in the information area.

Status Area

The status area is a small rectangular area that is usually located at the top of a window, below the menu bar. The status area is used to display information about the state of an object or the state of a particular view of an object.

To create a status area, first define an object `IStaticText statusLine` in the `AHelloWindow` class.

```
class AHelloWindow : public IFrameWindow,
                     public ICommandHandler
{
  public:
    AHelloWindow(unsigned long windowId);
    virtual ~AHelloWindow();
  protected:
    Boolean command(ICommandEvent& cmdEvent);
  private:
    IMenuBar      menuBar;
    IStaticText    hello;
    IInfoArea     infoArea;
    IStaticText    statusLine;
};
```

Next, modify the constructor of `AHelloWindow`. The status area is constructed when `AHelloWIndow` is created.

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow ( IFrameWindow::defaultStyle()
                 | IFrameWindow::minimizedIcon, windowId),
    menuBar(WND_MAIN,this),
    hello(WND_HELLO,this,this),
    infoArea(this),
    statusLine(WND_STATUS,this,this)
{
  statusLine.setText(STR_STATUS);
  addExtension(&statusLine,
    IFrameWindow::aboveClient,
    IFont(statusLine).maxCharHeight());

   ......
}
```

WND_STATUS is a control ID defined in the header file.
STR_STATUS, defined in the resource file, is a string resource ID that specifies a string to be displayed in the status area.

Basic Controls

The following sections provide information about creating and using the basic Presentation Manger controls with the User Interface Class Library. Included are discussions of the controls for static text, entry fields, push buttons, check boxes, radio buttons, and sliders.

Static Text Control

The `IStaticText` class allows you to create and manage static text in a window. Using this class, you can control the colors, position, and size of the text in the static text window.

Creating and Setting an IStaticText Instance: There are three ways to create an `IStaticText` instance:

1. From a control ID, parent and owner windows, rectangle, and style. Creates the specified static text control and an object for it.
2. From the ID of a static text control on a frame window. Creates the object for the specified static text control.
3. From the window handle of an existing static text control. Creates the object for the specified static text control.

To create an instance of the first type, you use the window ID, the parent window, and the owner window, as in the following example taken from the AHELLOW1.CPP sample file:

```
void main()                          //Main procedure with no parameters
{
  IFrameWindow * mainWindow=new        //Create our main window on the desktop
    IFrameWindow( x1   );             //  Pass in our Window ID
  IStaticText * hello=new IStaticText(   //Create static text control with
     x1 1 , mainWindow, mainWindow);     //   mainWindow as owner & parent
  hello->setText("Hello, World!");      //Set text in Static Text Control
  hello->setAlignment(                 //Set Alignment to Center in both
    IStaticText::centerCenter);         //  directions

  mainWindow->setClient(hello);         //Set hello control as client window
  mainWindow->setFocus();              //Set focus to main window
```

```
  mainWindow->show();                //Set to show main window

  IApplication::current().run();       //Get the current application and
                                    // run it
} /* end main */
```

Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for other ways of creating an instance.

Setting the Text: To set a text string in the `IStaticText` class, use the member function `setText`, as follows:

```
hello1->setText("This is an IStaticText");
```

The `setText` function is inherited from `ITextControl`.

Setting the Alignment: Using the `setAlignment` member function, you can position the text in nine places in a window. Figure 5 shows the nine locations for positioning text within a static text control.

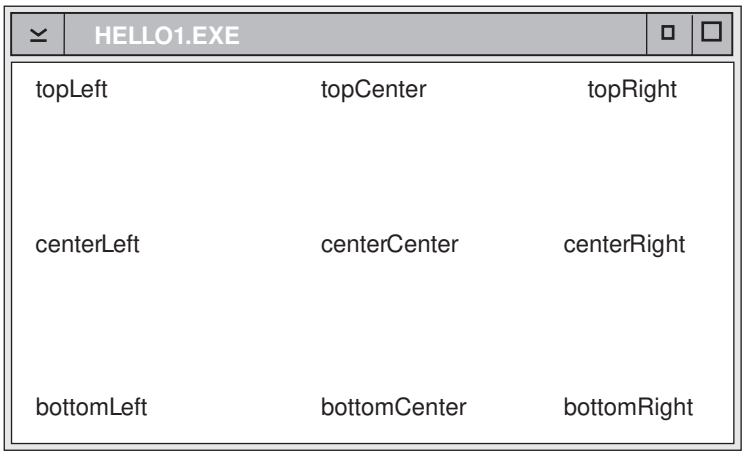


Figure 5. Aligning Text in a Window

The following example illustrates how to position text in a screen:

```
hello->setAlignment(              //Set alignment to center text
  IStaticText::topCenter);         // horizontally
```

Setting the Color: To set the foreground color used to draw the text, use the `setColor` function, as follows:

```
hello->setColor(IStaticText::foreground, IColor(IColor::cyan));
```

Entry Field Control

An entry field control is a control into which a user places text. There are three ways to create an instance of the `IEntryField` class:

1. From a control ID, parent and owner windows, rectangle, and style. Creates the specified entry field control and an object for it.
2. From the ID of a entry field control on a frame window. Creates the object for the specified entry field control.
3. From the window handle of an existing entry field control. Creates the object for the specified entry field control.

Figure 6 shows an example of an entry field control.

Figure 6. Example of an Entry Field Control

To create an `IEntryField` instance of the second type, first define a resource file and create a frame window that uses it, as in the following example from the ADIALOG4.DLG file (see “The Text Dialog Template” on page 204 for complete listing of ADIALOG4.DLG):

```
ENTRYFIELD "", DID_ENTRY, 8, 44, 114, 8, ES_MARGIN
```

You can manipulate the entry field using the `IEntryField` class. Supply the ID of the entry field (in this example, DID_ENTRY) and the window in which it is located. To set the text to the entry field, use the member function `setText`, as in the following example from the ADIALOG4.CPP file (see "The Text Dialog Source Code File" on page 199 for complete listing of ADIALOG4.CPP):

```
textField=new IEntryField(DID_ENTRY,  //Create entry field object using dialog
  this);                          //  entry field
textField->setText(textString);      //Set top current "Hello, World" text
```

After an event occurs, you can query the entry field for its contents using the member function `text`, as in the following statement from AWINDOW4.CPP:

```
 textValue = textField->text():       //Get text from frame entry field
```

Push Button Control

A push button is a button, labeled with text, graphics, or both, that represents an action that will be initiated when a user selects it. 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 manage the push button control window. To create an instance of this class, you can use one of the following constructors:

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

Generally, an application `ICommandEvent` event is generated when a push button is pressed. The application can change the style value to generate a help event or system `ICommandEvent` event by setting the following value into default style:

- `help`

  Causes a help event, instead of a command event, to be generated when the push button is pressed.

- `systemCommand`

  Causes an `ICommandEvent` system command event, instead of an application command, to be generated when the push button is pressed.

In Version 6 of the Hello World example, the member function `setupButtons` creates four push buttons, **Left**, **Center**, **Right**, and **Help**, at the bottom of the client window. The **Left** push button is created using the following statements:

```
IPushButton   * leftButton;
leftButton=new IPushButton(MI_LEFT, buttons, buttons, IRectangle(),
                     IPushButton::defaultStyle()
                     | IControl::tabStop);
leftButton->setText(STR_LEFTB);
```

The **Center** and **Right** push buttons are created with the same statements, except the push button ID and text shown on the item are different.

Because the **Help** push button generates a help event when it is pressed, the `help` style is added into default style.

```
helpButton=new IPushButton(MI_HELP, buttons, buttons, IRectangle(),
                      IPushButton::defaultStyle()
                     | IPushButton::help
                      | IControl::tabStop);
helpButton->setText(STR_HELPB);
```

By adding the `IControl::tabStop` into the default style, the user can use the Tab key to move the selection between these push buttons.

The command events generated by pressing the **Left**, **Center**, and **Right** push buttons are handled by the `command` member function. The help event generated by pressing the **Help** push button is handled by the `keysHelpId` member function.

For examples of `command` and `keyHelpId` member functions, refer to Chapter 16, “NLS and Advanced Functions” on page 251.

Check Box Control

A check box is a square box with associated text that represents a choice. When a user selects the choice, a “√” symbol appears in the check box to indicate that the choice is selected. By selecting the choice again, the user can clear the check box. A check box is used to set a choice in a group of choices that are not mutually exclusive.

The `ICheckBox` class allows you to create and manage a check box. The selection of a check box is processed by using the `ISelectHandler` class and adding the handler to either the check box or its owner window.

To create an instance of this class, you can use one of the following constructors:

```
ICheckBox(unsigned long Id, IWindow* parent, IWindow* owner,
        const IRectangle& initial= IRectangle(),
        const Style& style = defaultStyle());
ICheckBox(unsigned long Id, IWindow* parentDialog);
ICheckBox(IWindowHandle handle);
```

In the following example, the text associated with the check box is defined in the resource file as string text:

```
STRINGTABLE
  BEGIN
    STR_CHECK1  ,  "check box one"
    STR_CHECK2  ,  "check box two"
    STR_CHECK3  ,  "check box three"
  END
```

Each check box has a control ID, for example:

```
#define STR_CHECK1        x1  1
#define STR_CHECK2        x1  2
#define STR_CHECK3        x1  3
```

The following example defines three `ICheckBox` objects and constructs them in the `parentWindow`.

```
ICheckBox checkBox1( WND_CHECK1, &parentWindow, &ownerWindow );
checkBox1.setText(STR_CHECK1);
ICheckBox checkBox2( WND_CHECK2, &parentWindow, &ownerWindow );
checkBox2.setText(STR_CHECK2);
ICheckBox checkBox3( WND_CHECK3, &parentWindow, &ownerWindow );
checkBox3.setText(STR_CHECK3);
```

The `ISelectHandler` class processes item selection events for the `ICheckBox`, `IComboBox`, `IListBox`, and `IRadioButton` objects.

You can define a select handler class, which is inherited from the `ISelectHandler` class, to handle the check box selection. You must provide your own `selected` member function for the select handler class. Refer to the example given in “Radio Button Control” on page 38 for more information about how to define the select handler class and create the `selected` member function.

Radio Button Control

A radio button is a circle with text beside it. Radio buttons are used to display a fixed set of choices from which the user can select one. A group of radio buttons always contains at least two radio buttons.

The `IRadioButton` class allows you to create and manage the radio button control window. The `ISelectHandler` class processes the selection of a radio button and adds the handler to either the radio button or its owner window.

To create an instance of this class, you can use one of the following constructors:

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

The following example shows how to create two instances of this class:

```
IRadioButton radioBtBlack(WND_BLACKBT, &parentWindow, &ownerWindow);
radioBtBlack.setText(STR_BLACK);
IRadioButton radioBtWhite(WND_WHITEBT, &parentWindow, &ownerWindow );
radioBtWhite.setText(STR_WHITE);
```

WND_BLACKBT and WND_WHITEBT are control IDs given to each radio button.

These radio buttons are created in the `parentWindow`. A string for each radio button is defined as a string resource in the resource file:

```
STRINGTABLE
  BEGIN
    STR_BLACK,  "Black"
    STR_WHITE,  "White"
  END
```

You can use the `enableGroup` and `enableTabStop` member functions to set the group styles of a control. User the `select` member function to make the black button the default selected button, as shown below:

```
radioBtBlack.enableGroup().enableTabStop();
radioBtBlack.select();
```

The valid control styles are:

`group` Identifies the control as being the first in a group (arrow keys rotate through the group).
`tabStop` Identifies the control as one to which the user can tab.

You can define a select handler class, which is inherited from the `ISelectHandler` class, to handle the radio button selection, as follows:

```
class MySelectHandler : public ISelectHandler
{
  public:
    MySelectHandler() ;
  protected:
    selected( IControlEvent& evt );
  private:
   .........
};
```

To set the select handler to handle events from the selection of “Black” and “White” radio buttons, code the following:

```
selectHdr.handleEventsFor(&radioBtBlack);
selectHdr.handleEventsFor(&radioBtWhite);
```

You must provide your own `selected` member function for the select handler class, as in the following: as:

```
Boolean MySelectHandler::selected(IControlEvent& evt )
{
Boolean fProcess= false;
  switch (evt.controlId())
  {
  case WND_BLACKBT:
      ..............
      fProcess= true;
    break;
  case WND_WHITEBT:
      ..............
      fProcess= true;
    break;
  }
  return fProcess;
}
```

Slider Control

A progress indicator, or *slider*, is a control that represents a quantity and its relationship to the range of possible values for that quantity. A slider consists of a slider arm, slider shaft and, optionally, detents, tick marks, tick text, and slider buttons. Figure 7 shows the components of a slider control.

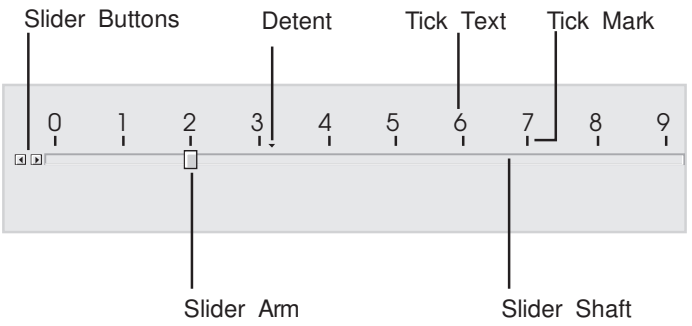


Figure 7. Slider components

Use the `ISlider` class to create a slider control, which enables users to set, display, or modify a value by moving the slider arm along the slider shaft.

The default style of `ISlider` positions the slider horizontally and centered in the window with tick marks and text above it. The slider arm is based on the left edge. You can also construct a slider with the tick marks and text below the shaft at various positions in the window, and you can position the slider vertically in the window.

The `ISlider` class is inherited from the `IProgressIndicator` class, which is a read-only version of the slider control. Typically, a progress indicator is used to display the percentage of a task that has been completed by filling in its shaft as the task progresses. Because the user cannot change the value represented by a progress indicator, a slider arm and slider buttons are not provided in a progress indicator. p. The progress indicator's shaft is filled with color as the arm moves.

The following example shows how to create a slider in the constructor of a subclass of `IFrameWindow`:

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
```

```
  : IFrameWindow (
    IFrameWindow::defaultStyle()
    | IFrameWindow::minimizedIcon,
    windowId)
{
  clientCanvas = new IMultiCellCanvas( WND_MCCANVAS, this, this );
  setClient( clientCanvas );

  infoArea = new IStaticText( WND_INFO, clientCanvas, clientCanvas );
  infoArea->setAlignment( IStaticText::centerCenter );
  infoArea->setText( STR_INFO );

  unsigned long numberOfTicks = 1 ;
  unsigned long tickSpacing=  ;

  mySlider = new ISlider( (unsigned long)ID_SLIDER,
                      clientCanvas, clientCanvas,
                      IRectangle(),
                      numberOfTicks,
                      tickSpacing,
                      ISlider::defaultStyle());

   mySlider->setTickLength(5);
   for (unsigned long z= ; z<1 ; z++)
       mySlider->setTickText(z, (char*)(IString(z)));

 unsigned long DetentId = mySlider->addDetent((unsigned long) 1 5) ;

  clientCanvas->addToCell(infoArea, 1, 1);
  clientCanvas->addToCell(mySlider ,1,2);
  clientCanvas->setColumnWidth(1,1  ,true);
  clientCanvas->setRowHeight(2,1  ,true);

  setFocus();
  show();
}
```

In the example, a multi-cell canvas is created and set to be the client area of this frame window. A static text control is created to show a message string. The slider is created with the default style, and tick marks, tick text, and detents are added to the slider.

Because tick marks are created with zero length and are, therefore, invisible, you must use the `setTickLength` member function to set the length of all tick marks on the slider scale. The `setTickText` member function sets the text associated with the tick at the specified index. The `addDetent` member function adds a detent to the slider at the pixel

offset from the home position specified, then returns a unique ID. This ID is required for removing a detent or querying its position.

The default style of a slider is horizontal. To change the style to vertical, add `IProgressIndicator::vertical` to the default style, as follows:

```
mySlider = new ISlider( (unsigned long)ID_SLIDER,
                        clientCanvas, clientCanvas,
                        IRectangle(),
                        numberOfTicks,
                        tickSpacing,
                        ISlider::defaultStyle()
                      | IProgressIndicator::vertical);
```

Several member functions are provided to perform the slider arm operation:

- `armPixelOffset`

  Returns the offset of the slider arm from the home position; the return value is the number of pixels.

- `armTickOffset`

  Returns the position of the slider arm; the return value is a tick number. Tick marks are numbered starting at zero.

- `moveArmToTick`

  Moves the slider arm to the specified tick number. Tick marks are numbered starting at zero.

- `moveArmToPixel`

  Moves the slider arm to a pixel offset relative to the home position.

For example, you can use the following statements to return the current slider arm position measured by tick offset, and then modify it to the result plus one.

```
unsigned long tickNumber= mySlider->armTickOffset ( );
tickNumber++ ;
mySlider->moveArmToTick ( tickNumber );
```

Figure 8 on page 43 shows the completed slider control.

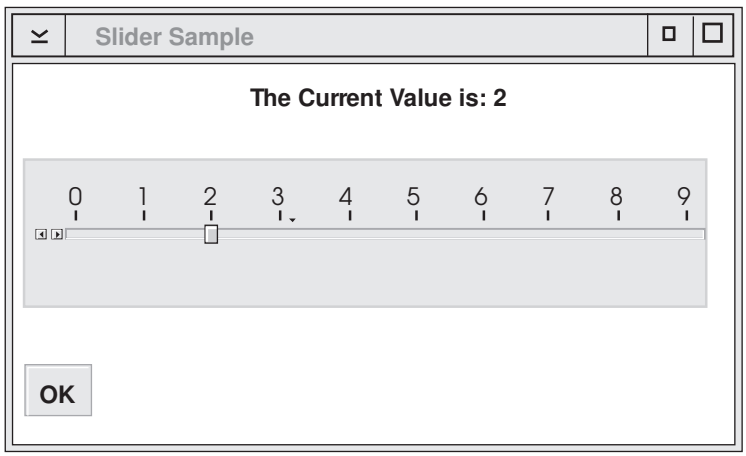


Figure 8. Slider Sample

Canvas Controls

The canvas classes provide a flexible way to build windows with multiple child controls. The various canvas classes provide different layout and sizing rules that enable you to build windows that contain fixed size areas, user sizeable areas, and scrollable areas. In addition, canvas controls allow you to control tabbing and cursor movement between child controls.

The following canvas classes are available:

- `ISplitCanvas`
- `ISetCanvas`
- `IMultiCellCanvas`
- `IViewPort`

Generally, you will build a complex window with a canvas control as the client area. This canvas may itself contain other canvas controls to provide the required layout.

Set and multicell canvases automatically size themselves to contain their child windows. This makes writing programs that are NLS compatible much easier, because the windows are sized dynamically when they are created while still retaining the same design. Because of the support for tabbing and cursoring between child controls, windows built using canvas classes provide an alternative to dialog boxes.

Split Canvas

A split canvas can contain two or more child controls. Each child control is placed in a pane. The panes are separated by moveable (default) or fixed split bars. A split canvas can have its split bars oriented vertically or horizontally.

A split canvas is best used to contain controls that can be resized to display more information. Examples of such controls are list boxes, containers, MLEs, and notebooks.

Note: Use the `noAdjustPosition` style on a list box control, when used within a 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, the child controls are arranged with the control created first in the leftmost pane. For a canvas with horizontal split bars, the control created first is placed in the top pane.

The following example shows how to create a window containing two split canvases. Each pane is occupied by a static text control.

1. The class declaration.

   The `ASplitCanvas` class is derived from the `IFrameWindow` class.

```
#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,                  // Note: the order of
                  vertCanvas;                 //  declaration is the order
    IStaticText   lText,                       //  that the windows are
                  rText,                      //  created
                  bText;
};
```

2. The constructor for the class.

   It creates a canvas with horizontal split bars and makes it the client area. A canvas with vertical split bars containing two static text controls is added to the top pane, and a static text control is added to the bottom pane.

```
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 );

  setFocus().show();              //Set focus and show window

} /* end ASplitCanvas :: ASplitCanvas(...) */
```

Figure 9 on page 47 shows the completed split canvas.

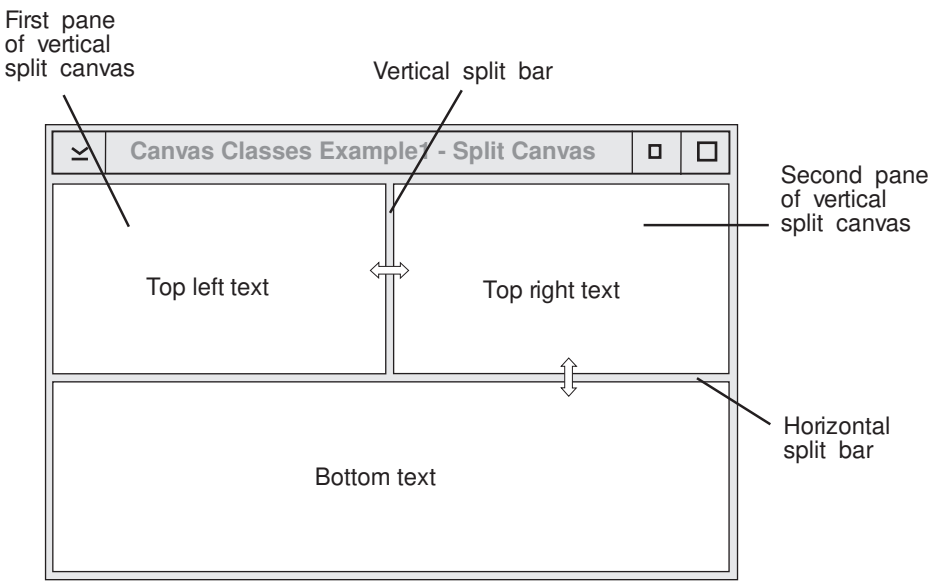


Figure 9. Split Canvas Example

Set Canvas

A set canvas arranges its child controls in either rows or columns. The User Interface Class Library uses the direction-independent term *deck* for either a row or column. You can arrange the deck or decks of a set canvas either horizontally or vertically. The set canvas attempts to place the same number of controls in each deck.

Each deck is large enough to contain the largest control in the deck. To do this, the canvas calls the `minimumSize` member function for each child control. For controls that have a size defined by the text they contain, such as push buttons and radio buttons, this default processing is normally sufficient. However, for controls that do not have a fixed size, such as notebooks, the control should set its minimum size overhiding `calcMinimizeSize` member function or calling the `setMinimumSize` member function before being added to the set canvas; otherwise, a default size is used.

The order in which you create the child controls determines both their position on the set canvas and the order in which tab and cursor keys switch focus between the controls. Several styles are available to control the orientation of the decks and the placement of controls within the decks. It is also possible to alter the spacing between controls and between the decks and the edge of the canvas.

The following example uses a split canvas as a client area. Two set canvases, each with seven radio buttons, are then added to the split canvas:

1. The class declaration.

   The `ASetCanvas` class is derived from the `IFrameWindow` class.

```
#include <iframe.hpp>                // IFrameWindow
#include <istattxt.hpp>               // IStaticText
#include <iradiobt.hpp>               // IRadioButton
#include <isetcv.hpp>                // ISetCanvas
#include <isplitcv.hpp>               // ISplitCanvas
#define  NUMBER_OF_BUTTONS  14

class ASetCanvas : public IFrameWindow
{
  public:                           //Define the public information
    ASetCanvas(unsigned long windowId); //Constructor for this class
   ~ASetCanvas();                    //Destructor for this class

  private:                          //Define private information
    ISplitCanvas    clientCanvas;
    IStaticText     status;
    ISetCanvas      vSetCanvas,
                   hSetCanvas;
    IRadioButton  * radiobut[NUMBER_OF_BUTTONS];
};
```

2. The constructor for the example uses a split canvas as the client area. A static text control is added as the top pane of the split canvas and below that two set canvases are added.

```
ASetCanvas::ASetCanvas(unsigned long windowId)
  : IFrameWindow( windowId )
  , clientCanvas( WND_SPLITCANVAS, this, this ,IRectangle(),
              ISplitCanvas::defaultStyle() | ISplitCanvas::horizontal)
  , 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);
  hSetCanvas.setPad(ISize(1 ,1 ));   //Set some space around buttons

  unsigned int i, mid = (NUMBER_OF_BUTTONS/2);
                                 //Create the first set of radio buttons
  for (i =   ; i < mid ; ++i)
    {
    radiobut[i]=new IRadioButton(WND_BUTTON+i, &vSetCanvas, &vSetCanvas);
    radiobut[i]->setText( STR_TEXT + i );
    }
  radiobut[ ]->enableGroup().enableTabStop();//Set tabStop and Group styles
  radiobut[ ]->select();                 //Select first button in group
```

```
                                    //Create the second set of radio buttons
  for (i = mid ; i < NUMBER_OF_BUTTONS ; ++i)
    {
    radiobut[i]=new IRadioButton(WND_BUTTON+i, &hSetCanvas, &hSetCanvas);
    radiobut[i]->setText(STR_TEXT + i);
    }
  radiobut[mid]->enableGroup().enableTabStop();//Set tabStop and Group styles
  radiobut[mid]->select();                  //Select first button in group

  radiobut[ ]->setFocus();            //Set focus to radio button one
  status.setText(STR_STATUS);          //Set status area text from resource

  show();                            //Show main window

} /* end ASetCanvas :: ASetCanvas(...) */
```

Figure 10 shows the set canvas created using this code.

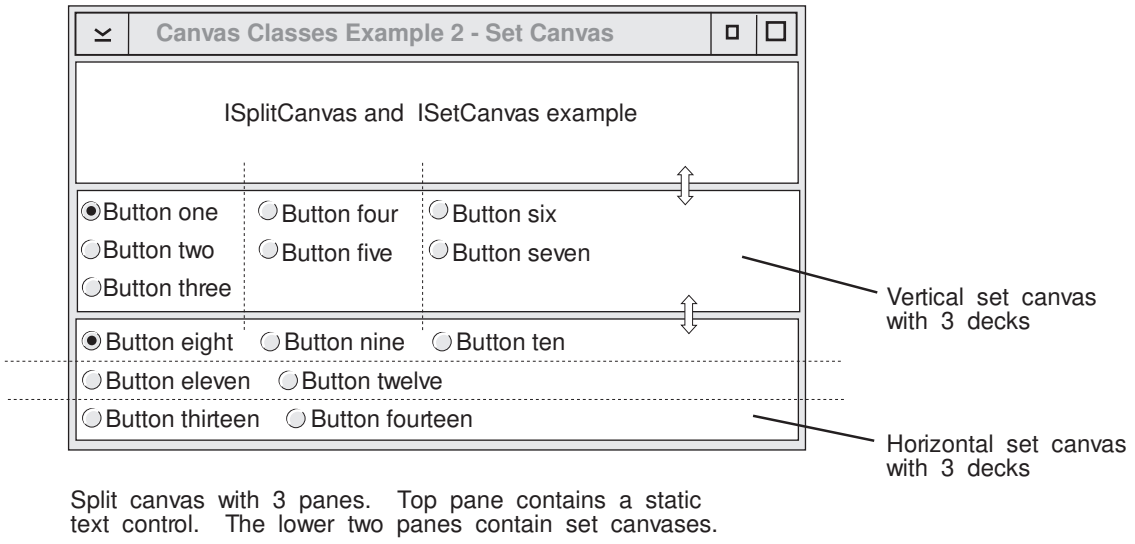


Figure 10. Set Canvas Example

Multicell Canvas

A multicell canvas consists of a grid of rows and columns. Child controls are placed on the canvas by specifying the starting cell and the number of contiguous rows and columns that they are allowed to span. Cells in the grid can be referred to by giving the column and row value, with the top left cell coordinate being (1,1). The actual number of rows and columns in the canvas is the highest row and column value used. In the following example, a radio button is placed at (4,5) and a push button at (2,7). Therefore the canvas will be at least 4 columns and 7 rows.

The initial size of a row or column is determined by the size of the largest control in that row or column. By default, rows and columns are of fixed size. If necessary, they can be made expandable, in which case sizing the canvas causes them to resize.

Rows and columns can be left empty to provide spacing between child controls. If you do this, you need to explicitly specify the size of the empty rows and columns.

The following example code shows how to create a window containing a multicell canvas. The canvas contains two check boxes, two radio buttons, three static text controls, and a single push button.

1. The class declaration.

   The `AMultiCellCanvas` class is derived from the `IFrameWindow` class.

```
#include <iframe.hpp>                    // IFrameWindow
#include <istattxt.hpp>                   // IStaticText
#include <ipushbut.hpp>                   // IPushButton
#include <iradiobt.hpp>                   // IRadioButton
#include <icheckbx.hpp>                   // ICheckBox
#include <imcelcv.hpp>                    // IMultiCellCanvas
class AMultiCellCanvas : public IFrameWindow
{
  public:
    AMultiCellCanvas(unsigned long windowId);

  private:
    IMultiCellCanvas    clientCanvas;
    IStaticText        status,
                    title1,
                    title2;
    ICheckBox          check1,
```

```
                    check2;
    IRadioButton      radio1,
                    radio2;
    IPushButton       pushButton;
};
```

2. The constructor for the multicell canvas window.

   It creates a multicell canvas and makes it the client area. The other controls are then placed on the canvas using the `addToCell` member function.

```
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 multicell canvas the client
  setClient( &clientCanvas );
                                // set status area text
  status.setAlignment( IStaticText::centerCenter );
  status.setText( STR_STATUS );

  title1.setAlignment( IStaticText::centerLeft );   // set text and alignment
  title1.setText( STR_TITLE1 );

  title2.setAlignment( IStaticText::centerLeft );   // set text and alignments
  title2.setText( STR_TITLE2 );

  check1.setText( STR_CHECK1 );                  // set checkbox text
  check2.setText( STR_CHECK2 );
  radio1.setText( STR_RADIO1 );                  // set radio button text
  radio2.setText( STR_RADIO2 );

  pushButton.setText( STR_PUSHBUT );

  radio2.select();                    // pre-select one radio button
  check1.enableGroup().enableTabStop();  // set tabStop and Group styles
  radio1.enableGroup().enableTabStop();
  pushButton.enableGroup().enableTabStop();

  clientCanvas.addToCell(&status  , 1, 1, 4, 1);    // add controls to canvas.
  clientCanvas.addToCell(&title1  , 1, 3, 2, 1);    // the canvas runs from
  clientCanvas.addToCell(&title2  , 3, 3, 2, 1);    // 1,1 to 4,7
```

```
    clientCanvas.addToCell(&check1  , 2, 4);          // exactly one row and
    clientCanvas.addToCell(&check2  , 2, 5);          // one column are
    clientCanvas.addToCell(&radio1  , 4, 4);          // expandable, as this
    clientCanvas.addToCell(&radio2  , 4, 5);          // allows the canvas to
    clientCanvas.addToCell(&pushButton , 2, 7);       // fill the whole client.

                              // make row 2 2  pixels high and expandable
    clientCanvas.setRowHeight(2, 2 , true);
                              // make row 6 4  pixels high
    clientCanvas.setRowHeight(6, 4 );
                              // make column 4 4  pixels wide and expandable
    clientCanvas.setColumnWidth(4, 4 , true);

    check1.setFocus();                     // set focus to first checkbox
    show();                               // show main window

  } /* end AMultiCellCanvas :: AMultiCellCanvas(...) */
```

Figure 11 shows the completed multicell canvas.

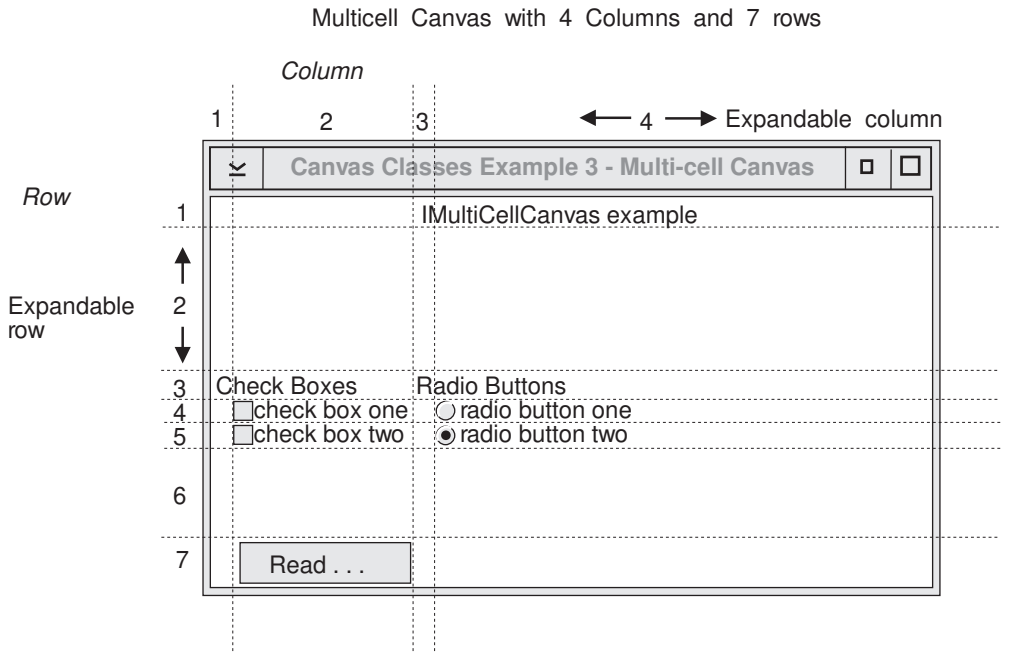


Figure 11. Multicell Canvas Example

Viewport

A viewport differs from other canvas types in that it allows only one child control. The size of the child control is not altered when the viewport is resized. If the viewport is smaller than the child control, scroll bars are added to the viewport. The user can then use the scroll bars on the viewport to view different parts of the child control. The child control does not need to provide any code to handle scrolling.

If several controls are required inside a viewport, they should be placed into another type of canvas, which can then be made the child of the viewport.

The following example code shows how to create a window containing a viewport. The viewport has a single bit-map control inside it.

1. The class declaration.

   The `AViewPort` class is derived from the `IFrameWindow` class.

```
#include <iframe.hpp>                 // IFrameWindow
#include <ivport.hpp>                 // IViewPort
#include <ibmpctl.hpp>                // IBitmapControl
class AViewPort : public IFrameWindow
{
  public:                           // define the public information
    AViewPort(unsigned long windowId);   // constructor for this class

  private:                          // define private information
    IViewPort      clientViewPort;
    IBitmapControl  bitmap;
};
```

2. The constructor for the viewport window.

   It creates a viewport control and sets it as the client area. The bit-map control is then made a child of the viewport.

```
#include <ireslib.hpp>              // IResourceId class
AViewPort :: AViewPort(unsigned long windowId)
  : IFrameWindow( windowId )
  , clientViewPort( WND_VIEWPORT, this, this )
  , bitmap( WND_BITMAP, &clientViewPort, &clientViewPort
        , IResourceId(BMP_ID) )
{
                                  // make viewport the client
  setClient( &clientViewPort );

  setFocus().show();              // set focus and show window
```

```
} /* end AViewPort :: AViewPort(...) */
```

Figure 12 shows the viewport canvas created using this code.

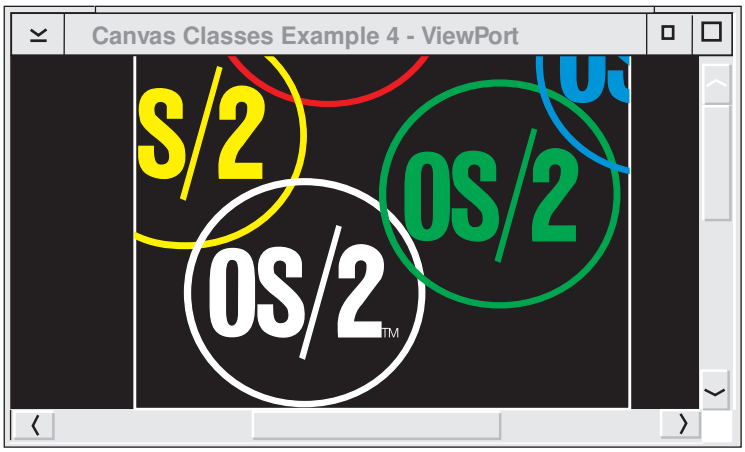


Figure 12. Viewport Canvas Example

Styles

Every window provided by the User Interface Class Library has a number of different styles that affect the appearance and behavior of the window. These styles are encapsulated in style objects. The style classes operate within the scope of the window class that they affect. Generic styles are defined in `IWindow` and `IControl`. Windows that are subclasses of these can combine their own styles with those of `IWindow` and `IControl`.

Each window class maintains its own default style. This default style object can be accessed using the static member function `defaultStyle` and set using the static member function `setDefaultStyle`. Each class also maintains a style object called `classDefaultStyle` that corresponds to the initial setting of `defaultStyle`.

Style Objects

Style objects can be combined using the | (bitwise OR) operator. Styles can also be removed from a style object by creating a negated-style object using the ~ (logical negation) operator and then using the & (logical AND) operator.

1. Combine two styles

   The following example creates a list box style object that can be used to construct a multiple-selection list box.

   ```
   IListBox::Style lbStyle = IListBox::defaultStyle()
                           | IListBox::multipleSelect;
   ```

2. Remove a style

   The following example creates a list box style object that can be used to construct a list box without a horizontal scroll bar:

   ```
   IListBox::Style lbStyle = IListBox::defaultStyle()
                           & ~IListBox::horizontalScroll;
   ```

Setting Window Styles

The User Interface Class Library provides three ways to create a window with a specific style:

1. Create a window using a constructor which accepts the style as a parameter
2. Create a window with the default style and change it using member functions of the window
3. Change the default style for the window class and then construct the window

All windows provide one or more constructors that accept a style object as one parameter. A style object can only be constructed from existing style objects. Additionally, you can combine style objects using the | (logical OR) operator.

The following example shows how to create an entry field control with a style that is a combination of styles taken from `IWindow`, `IControl` and `IEntryField`.

```
IEntryField  entryField( ID_EF1, parent, owner,
                    IRectangle(1 , 1 , 1  , 2 ),
                    IWindow::visible    |
                    IControl::tabStop   |
                    IControl::group     |
                    IEntryField::margin |
                    IEntryField::autoScroll );
```

Alternatively, the style object can be explicitly constructed and passed as a parameter:

```
IEntryField::Style efStyle = IWindow::visible    |
                        IControl::tabStop   |
                        IControl::group     |
                        IEntryField::margin |
                        IEntryField::autoScroll ;
IEntryField  entryField( ID_EF1, parent, owner,
                    IRectangle(1 , 1 , 1  , 2 ),
                    efStyle );
```

Each window class maintains its own default style. This default style object can be accessed using the static member function `defaultStyle`. This simplifies the preceding example to:

```
IEntryField  entryField( ID_EF1, parent, owner,
                    IRectangle(1 , 1 , 1  , 2 ),
                    IEntryField::defaultStyle()  |
                    IControl::tabStop          |
                    IControl::group );
```

Use the static member function `setDefaultStyle`. to set the default style for a class. For example:

```
IEntryField::Style efStyle = IEntryField::defaultStyle()  |
                       IControl::tabStop          |
                       IControl::group    ;
IEntryField::setDefaultStyle(efStyle);
IEntryField  entryField( ID_EF1, parent, owner,
                    IRectangle(1 , 1 , 1  , 2 ) );
```

To set the style of a window after creating it, use specific member functions provided by each class. The example now becomes:

```
IEntryField  entryField( ID_EF1, parent, owner,
                    IRectangle(1 , 1 , 1  , 2 ) );
entryField.enableGroup();             // member function of IControl
entryField.enableTabStop();           // member function of IControl
entryField.enableAutoScroll();        // member function of IEntryField
```

For a complete list of available styles, see the *IBM C/C++ Tools: User Interface Class Library Reference*.

Cursors

The User Interface Class Library provides cursor classes to iterate through collections of items. Window classes that can contain one or more items generally provide a nested cursor class. Cursors are usually constructed by providing the window to iterate over (look at each item).

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 be pointing at an item in the list will validate 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 over changes by the addition or removal of items, the cursor becomes invalid and 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).

All cursor classes provide member functions to move through the items, either forward or backward, and to add items after the cursor position.

In some cases, you may want to construct a cursor that iterates only over items with a particular property. For example, the constructor for a list box cursor takes a second parameter that determines whether the cursor returns all items in the list box or just the selected items.

The following example shows how to iterate through all selected items in a multiple-selection list box:

```
IListBox listbox( ID_LB, parent, owner, IRectangle(),
                  IListBox::defaultStyle() | IListBox::multipleSelect );
/*  ...  add items to listbox  ... */
IListBox::Cursor lbCursor(listbox);
for (lbCursor.setToFirst(); lbCursor.isValid(); lbCursor.setToNext())
   {
   IString  str(listbox.elementAt(lbCursor));     //Return item at cursor
   unsigned long ul = lbCursor.asIndex();         //Return zero-based index
   /* ... process string or index ... */
   }
```

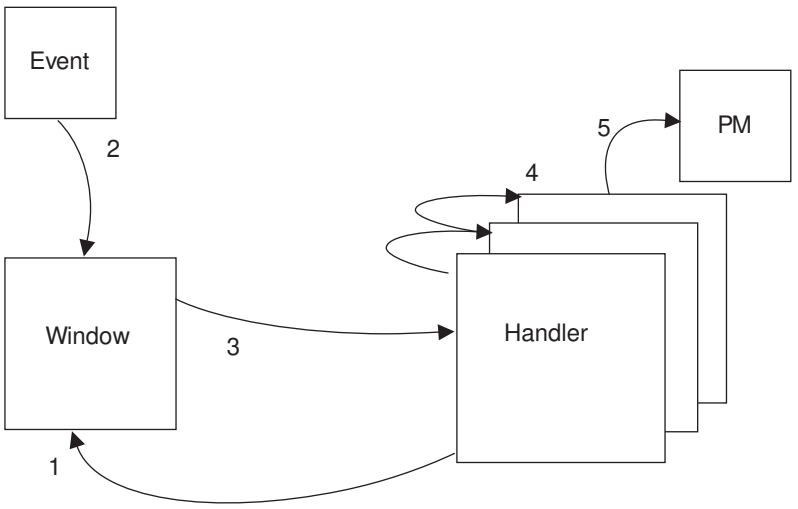
Chapter 4. Handlers and Events

Handlers and Events

The User Interface Class Library uses *handlers* and *events* to encapsulate the message architecture of OS/2 Presentation Manager in an object-oriented way. Presentation Manager messages are encapsulated in event objects, which are passed to the window or control that had the event. The window then invokes the handlers that have been attached to the window, passing the event object as a parameter. The handlers are called sequentially with the most recently added handler being invoked first. A handler must return a `Boolean` value to indicate whether processing for the event is complete. A return value of true indicates to the window that all processing for the event has been completed. The event is not routed to any other handler attached to the window or passed to the default PM window procedure for the window. Events that are not processed by any handler are processed by the window procedure for the underlying PM window.

Figure 13 on page 62 shows the main relationships between window, event, and handler classes.

© Copyright IBM Corp. 1993



1 - handler registered with window
2 - event routed to window
3 - window dispatches event
4 - event passed on until processed or
5 - event passed to PM for default processing

Figure 13. Windows, events, and handlers

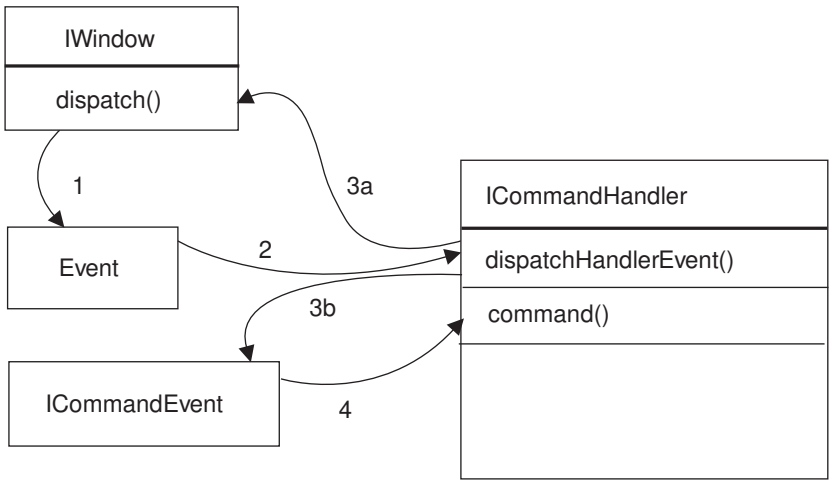
The distinction between window classes and handler classes provided by the User Interface Class Library allows an application to separate its own event handling logic from the rest of the application. This enables the reuse of event handling logic. For example, a handler to verify telephone numbers can be reused wherever there is an entry field which accepts telephone numbers.

Handlers

Each handler class has one or more virtual functions that are called to process the event. When an application processes events, it normally subclasses a handler class and overrides the virtual function to provide its own application-specific logic.

Note: Handlers need to return from the virtual functions within 1/10 second to avoid locking up the system by delaying the PM message processing.

Figure 14 on page 63 shows how the `ICommandHandler` works. All handler classes contain a `dispatchHandlerEvent` function to determine whether the handler needs to process the event or return it. If the event needs processing, it creates the appropriate event object and calls the appropriate virtual function to process the event.



1 - window creates event
2 - IEvent passed to ICommandHandler
3a - IEvent not processed
3b - ICommandEvent generated
4 - ICommandEvent processed by command()

Figure 14. Processing within the ICommandHandler

The following table presents some of the more common events for which you might want to provide handlers. It relates the type of event, the handler for that event, and the member function in the handler class that the application must override in order to provide its own logic. The *IBM C/C++ Tools: User Interface Class Library Reference* contains a description of all handler classes and member functions.

Figure 15. Handler Classes and Their Member Functions

Type of Event	Generated by	Event Class	Handler Class	Function
Command event	Menu selection, push button, accelerator key	ICommandEvent	ICommandHandler	command
System command event	Menu selection, push button, accelerator key	ICommandEvent	ICommandHandler	systemCommand
Edit event	Entry field, combination box, MLE, slider	IControlEvent	IEditHandler	edit
Gain focus or lose focus	Entry field, combination box, MLE, slider, container, spin button	IControlEvent	IFocusHandler	getFocus or lostFocus
Keyboard entry	Entry field, combination box, MLE or other input focus control	IKeyboardEvent	IKeyboardHandler	keyPress, scanCodeKeyPress, virtualKeyPress, characterKeyPress, key
Paint area event	All controls	IPaintEvent	IPaintHandler	paintWindow
Resize event	All controls	IResizeEvent	IResizeHandler	windowResize
Item selected	List box, combination-box, container, check box, radio button	IControlEvent	ISelectHandler	selected
Enter pressed when item selected, or double-click on item	List box, combination-box, container	IControlEvent	ISelectHandler	enter
Menu about to be shown	Menu bar	IMenuEvent	IMenuHandler	menuShowing
Context menu of container item requested	Container	IMenuEvent	ICnrMenuHandler	makePopupMenu

Events

The `IEvent` class acts as the base class for the more specialized event classes. It provides general member functions to extract the message ID and message parameters. The subclasses of `IEvent` generally add more specialized functions for extracting information specific to that type of event.

The following table shows some of the more common event classes, and some of the functions they contain to extract event information. Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for a complete list of event classes and member functions.

Figure 16. Event classes and accessor functions

Event Class	Accessor Function	Description of Return Value
IEvent	window	The IWindow object pointer
IEvent	handle	IWindowHandle of the window
IEvent	eventId	ID of the event
IEvent	parameter1	IEventData containing first event parameter
IEvent	parameter2	IEventData containing second event parameter
ICommandEvent	source	An enumeration type which gives the type of control
ICommandEvent	commandId	The ID of the control that caused the event
IControlEvent	controlId	The ID of the control that caused the event
IControlEvent	control	Pointer to the control that caused the event
IKeyboardEvent	character	Single-byte character code (exception thrown if DBCS)
IKeyboardEvent	mixedCharacter	IString containing character (may be DBCS)
IKeyboardEvent	virtualKey	An enumeration type which gives the virtual key event
IMenuEvent	menuItemId	The ID of the selected menu Item
IMenuEvent	mousePosition	Position of mouse at the time the event occurred
IPaintEvent	presSpaceHandle	The handle of the presentation space to use for any drawing
IPaintEvent	rect	The screen rectangle that needs updating

The `IEvent` class provides a member function, `setResult`, for those events that require a value to be returned.

Writing a Handler

In general, handling an event can be divided into four distinct stages.

1. Determine which handler class processes the event
2. Subclass the handler class and override the event handling functions
3. Create an instance of your subclass
4. Attach the instance to the window

The Hello World applications have several event handlers. The following code is taken from Hello World Version 3 (see Chapter 13, "Event Handling and Menu Bars" on page 173) and shows how to process user menu selection.

1. Determine which handler class processes the event

   Selecting a menu results in a command message being sent to the frame window and the client window. The handler class for this type of event is `ICommandHandler`.

2. Subclass the handler class and override the event handling function

   The Hello World application uses multiple inheritance to provide a class `AHelloWindow` which inherits from both `IFrameWindow` and `ICommandHandler`. The class `ICommandHandler` has a virtual function `command` which is invoked to process command events. The class `AHelloWindow` overrides this function in order to provide its own command handling. The following extract from AHELLOW3.HPP shows the class declaration.

```
class AHelloWindow : public IFrameWindow,
                     public ICommandHandler
{
  public:                              // define the public information
    AHelloWindow(unsigned long windowId); // constructor for this class

  protected:                            // define protected member
    Boolean command(ICommandEvent& cmdEvent);

  /*  ... rest of class declaration ...   */

};
```

   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.

```
Boolean AHelloWindow :: command(ICommandEvent & cmdEvent)
{
  switch (cmdEvent.commandId()) {      // get command id
    case MI_CENTER:                  // process center command
  /*  ... process center menu item ... */
     return(true);                   // return command processed
     break;
    case MI_LEFT:                    // code to process left command
  /*  ... process left menu item ... */
     return(true);                   // return command processed
     break;                         //
    case MI_RIGHT:                   // code to process right command
  /*  ... process left menu item ... */
     return(true);                   // return command processed
     break;                         //
  }

  return(false);                     // return command not processed
} /* end HelloWindow :: command(...) */
```

3. Create an instance of your subclass.

   Since the window is its own command handler, creating the window creates an instance of the handler. In the case where a separate handler class has been defined, it would be necessary to create an instance of it. Normally this would be done during the constructor for the window.

4. Attach the instance to the window.

   The base class `IHandler` provides a member function `handleEventsFor` to attach a handler to a window. In the Hello World example, the handler is attached during the constructor for the window.

```
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow (                     // call IFrameWindow constructor
    IFrameWindow::defaultStyle()      // use default plus
    | IFrameWindow::minimizedIcon,    // get minimized icon from  RC file
    windowId)                          // main window ID
{
  /*  ... constructor code ... */

  handleEventsFor(this);             // set self as command event handler

} /* end AHelloWindow :: AHelloWindow(...) */
```

Chapter 5. Data Types and Attributes

Managing Character Data

The `IString` class contains member functions that enable you to perform a variety of data manipulation and management activities.

You can perform the following tasks with the `IString` class:

- Perform stream I/O
- Query string characteristics
- Test the contents of the string
- Compare strings using overloaded operators
- Convert string
- Edit strings
- Manipulate strings using concatenation, copy, and alignment operators.

Stream I/O

You can read and write an `IString` instance using the operators << and >>. The following example shows how to do this:

```
#include <istring.hpp>
#include <iostream.h>
void main()
{
IString
   s1="Enter a letter = ",
   s2;
  cout << s1 << endl;
  cin >> s2 ;
} /* end main */
```

Following are the results of running this program:

```
[C:\]t1
Enter a letter =
a
[C:\]
```

© Copyright IBM Corp. 1993

Accessors

The `IString` class provides accessors that allow you to analyze various elements of a string. Some examples are:

`size` Returns the size of the string.
`Substring` Returns one part of the string.
`Operator[]` Returns the value of the *n*th position in the string.

In the following example, the highlighted commands show examples of how to use these accessors to determine the size of the string:

```
#include <istring.hpp>
#include <iostream.h>
void main()
{
  IString
    s1("string"),
    s2;
  cout << " The size of s1 is " << s1.size() << endl;
  cout << "And the 5th element is " << s1[5] << endl;
  cout << "And the first three characters are " << s1.subString(1,3) << endl;
} /* end main */
```

The results of this program are as follows:

```
[C:\]t2
 The size of s1 is 6
And the 5th element is n
And the first three characters are str
[C:\]
```

Testing

Use the `IString` member functions to test for certain conditions or characteristics of a string. For example:

`isAlphanumeric`	Returns true if all characters are in ('A-Z,'a-z','0-9').
`isAlphabetic`	Returns true if all characters are in ('A-Z,'a-z').
`isDigits`	Returns true if all characters are in ('0-9').
`isLowerCase`	Returns true if all characters are in ('a-z').
`isUppercase`	Returns true if all characters are in ('A-Z').
`isControl`	Returns true if all characters are in (0x00-0x1F, 0x7F).

The following examples show how to identify one string:

```
#include <istring.hpp>
#include <iostream.h>
void main()
{
   IString s,s1;

   cin >> s;
   if (s.isDigits())
   {cout << "the string " << s << " contains  only numbers" << endl;}
   else
   {
     if (s.isAlphabetic())
     {
      if (s.isLowerCase())
      { cout << "the string " << s << " contains  only lowercase characters" << endl;}
      else
       if (s.isUpperCase())
       {cout << "the string " << s << " contains  only uppercase characters" << endl;}
       else
       {cout << "the string " << s << " contains  only mixed Alphabetic characters" << endl;}
     }
     else
     {
      if (s.isAlphanumeric())
        { cout << "the string " << s << " contains  only Alphanumeric characters" << endl;}
     else
      { cout << "the string " << s << " contains  some unusual characters" << endl; }
     }
   }
} /* end main */
```

The results of this program are as follows:

```
[C:\]t3
ABC
the string ABC contains  only uppercase characters

[C:\]t3
abc
the string abc contains  only lowercase characters

[C:\]t3
Abc
the string Abc contains  only mixed Alphabetic characters

[C:\]t3
12a
the string 12a contains  only Alphanumeric characters

[C:\]t3
#@%
the string #@% contains  some unusual characters

[C:\]
```

Comparison

The `IString` class includes a full set of comparison operators for comparing an `IString` to another `IString` or to a literal character string:

`==` Returns true if the strings are identical.
`!=` Returns true if the strings are not identical.
`<` Returns true if the first string is less than the second, applying the standard collating scheme(memcmp).
`<=` Equivalent to `(string1 < string2)||(string1 == string2).`
`>` Equivalent to `!(string1 <= string2).`
`>=` Equivalent to `!(string1 < string2).`

The following example compares two strings to determine if they are equal:

```
#include <istring.hpp>
#include <iostream.h>
void main()
{

IString s("Name"), s1;

cin >> s1;

    if (s1 == s)
     cout << s1 << " is equal to " << s << endl;
    else
     if (s1 != "name")
      cout << s1 << " is not expected" << endl;
     else
      cout << " You forgot the first letter is capital" << endl;
} /* end main */
```

The results of the program are as follows:

```
[C:\]t4
12
12 is not expected

[C:\]t4
name
 You forgot the first letter is capital

[C:\]t4
Name
Name is equal to Name

[C:\]
```

Conversion

The `IString` class provides member functions to convert strings into other values. For example:

`asInt` Converts a string into a long integer.
`asDouble` Converts a string into a double.

Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for additional conversion member function.

The following example converts into a long integer:

```
#include <istring.hpp>
#include <iostream.h>
void main()
{

IString s1("111 1");
int n1;

 n1=s1.asInt();
 n1+=1;
 s1=n1;
 cout << s1 << endl;

} /* end main */
```

Here are the results of this program:

```
[C:\]t5
111 2

[C:\]
```

Modifying and Aligning

The `IString` class provides member functions for modifying and aligning text strings. The following member functions are available:

`change` Changes occurrences of an argument to an argument replacement string.

`center` Centers the receiver within a string of a specified length.

`leftJustify` Left justifies the receiver within a string of a specified length.

`rightJustify` Right justifies the receiver within a string of a specified length.

`upperCase` Translates all lowercase letters in the receiver to uppercase.

`lowerCase` Translates all uppercase letters in the receiver to lowercase.

The following examples show how to replace one string with another, change the text alignment, and translate text from lowercase to uppercase:

```
#include <istring.hpp>
#include <iostream.h>
void main()
{


IString s("text"),s1,s2,s3,s4, s5("this is a test");
  s4 = s3 = s2 = s1 = s;
  cout << " | " << s1.center(1 ,'+') << " | " << endl;
  cout << " | " << s2.leftJustify(1 ,'<') << " | " << endl;
  cout << " | " << s3.rightJustify(1 ,'>') << " | " << endl;
  cout << " | " << s4.upperCase().center(1 ,' ') << " | " << endl;
  cout << " | " << s4.upperCase().center(1 ,' ') << " | " << endl;
  cout << " | " << s5.change("this","these").change("is","are")
  .change("test","tests").change("a ","",8,1) << " | " << endl;

} /* end main */
```

Here are the results:

```
[C:\]t6
 | +++text+++ |
 | text<<<<<< |
 | >>>>>>text |
 |    TEXT    |
 |    TEXT    |
 | these are tests |

[C:\]
```

Manipulation

The `IString` class provides operators to manipulate text in a variety of ways:

= Assigns the following string to a receiver.
~ Performs bitwise negation.
+ Concatenates two strings.
+= Concatenates and replaces.

The following example shows how to concatenate two strings:

```
#include <istring.hpp>
#include <iostream.h>
void main()
{

IString s("1"), s1("2"), s2;

   s2 = s + s1;
   if (s2 != "12") cout << " Something is wrong " << endl;
   else cout << " I expected that " << endl;

} /* end main */
```

The results of this program are as follows:

```
[C:\]t7
 I expected that

[C:\]
```

Fonts

Use the `IFont` class to set and change all characteristics of the fonts you use in your applications.

To create a system default font, use the following statement:

```
IFont curFont();
```

To create a font of a specific name and point size, use the following statement:

```
IFont curFont("Helv",1 );
```

The following example creates a font using the current font from the hello window:

```
IFont curFont(&hello);
```

where `hello` is the `IWindow` instance.

The `IFont` class includes member functions that enable you to set or change a variety of font characteristics, for example:

`setName` Set the name of the font.

`setPointSize` Set the font's point size.

`setBold` Use a bold font.

`setItalic` Use an italic font.

`setUnderscore` Use a font that underscores.

`setStrikeout` Use a font that contains character strike-overs.

`setOutline` Use an outline font.

`setAllEmphasis` Use emphasis on all text.

You can set the font of almost all objects in the User Interface Class Library using the member function `setFont`. Because this member function is defined in the `IControl` class, it is inherited for classes several classes, including `IStaticText`, `IEntryField`, `IPushButton`, and `IMultiLineEdit`. Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for the classes derived from `IControl`.

The following example shows how to create a font with a specific name and point size, and then change the point size of different text strings:

```
#include <ifont.hpp>
  .
IFont  Fonts("Helv",8);
  .
  .
 title1 = new IStaticText( WND_TITLE1, clientCanvas, clientCanvas );
 title1->setAlignment( IStaticText::centerLeft );
 title1->setText( STR_TITLE1 );
Fonts.setPointSize(12);
title1->setFont(Fonts);
  .
  .
 check1 = new ICheckBox( WND_CHECK1, clientCanvas, clientCanvas );
 check1->setText( STR_CHECK1 );
Fonts.setPointSize(2 );
check1->setFont(Fonts);
  .
```

To test the font statements, include the highlighted lines into the AMCELCV.CPP file.

Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for information about member functions used to query the appearance of a font.

Part 2. Beyond the Basics

© Copyright IBM Corp. 1993

Chapter 6. Advanced Controls

Multiple-Line Entry Field Control

Use the `IMultiLineEdit` class to create a multiple-line entry (MLE) field control. The `IMultiLineEdit` class member functions enable you to display text files with horizontal and vertical scrolling, read a file into an MLE and save it from an MLE, or perform basic editing tasks, such as cut, copy, clear, discard, paste marked lines and move to top or bottom of the MLE.

Figure 17 shows the hierarchy of the `IMultiLineEdit` class.

```
IBase
    IVBase
        IWindow
            IControl
                ITextControl
                    IMultiLineEdit
```

Figure 17. Hierarchy for IMultiLineEdit Class

Creating an IMultiLineEdit Instance

To create an instance of this class, you include the ID of a specified multiple-line entry field control, the parent and owner windows, an `IRectangle` instance, and one or more styles. Styles are available to define such features as scrolling text, wrapping words, adding a border, and making the field read-only. Refer to *IBM C/C++ Tools: User Interface Class Library Reference* for further information about creating an instance of this class and the styles used with this class.

© Copyright IBM Corp. 1993

The following example creates an instance of an MLE that uses the default style and includes horizontal scrolling:

```
 .
 .
AEditorWindow :: AEditorWindow(unsigned long windowId)
  : IFrameWindow (                  //Call IFrameWindow constructor
    IFrameWindow::defaultStyle()       //Use default plus
    | IFrameWindow::minimizedIcon,     //Get minimized icon from  RC file
    windowId)                        //Main window ID
{
  mtextfield = new IMultiLineEdit( DID_MLE, this, this,IRectangle(),
                                   IMultiLineEdit::defaultStyle() |
                                   IMultiLineEdit::horzScroll);
 .
  setClient(mtextfield);
 .
 .
```

where DID_MLE is the ID of an MLE defined in the resource file.

The preceding command creates an instance of an `IMultiLineEdit` class. When it is set in a client window, it looks like Figure 18.

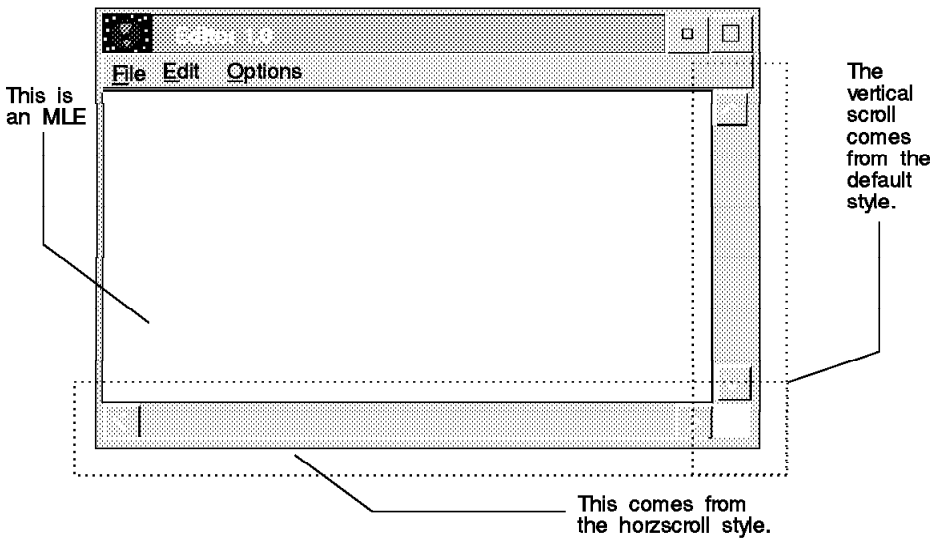


Figure 18. Multiple-line Entry Field Control

Loading and Saving a File

Three member functions allow you to manage files and MLEs:

- importFromFile
- exportToFile
- exportSelectedTextToFile

Using these functions, you can load a file into an MLE, save a file from an MLE, or save marked text in an MLE into a file. The following example illustrates how to load a file into an MLE:

```
.
.
   filename=fd->fileName();          //
   if (filename.size())             //Has filename been specified?
   {                               //
  mtextfield->importFromFile(filename.asString());
   mtextfield->setCursorAtLine( );
    } /* endif */                   //
 } /* endif */                     //
.
.
```

Refer to *IBM C/C++ Tools: User Interface Class Library Reference* for descriptions of these member functions.

Positioning the Cursor

Using the cursor position, you can position a cursor in a specific line of an MLE or in a specific character position, add or remove lines, or ask for the number of lines in an MLE. These are the member functions available for line operations:

- addLine
- addLineAsLast
- removeLine
- setTop
- setCursorAtLine
- setCursorAt
- top
- cursor
- numberOfLines
- visibleLines

The following example shows how to position the cursor at the first line of an MLE into which a file has been imported:

```
.
.
   filename=fd->fileName();           //
   if (filename.size())              //Has Filename been specified?
   {                              //
  mtextfield->importFromFile(filename.asString());
  mtextfield->setCursorAtLine( );
   } /* endif */                     //
 } /* endif */                      //
.
.
```

Figure 19 shows the file that has been imported into the MLE. Notice that the cursor is positioned on the first line.

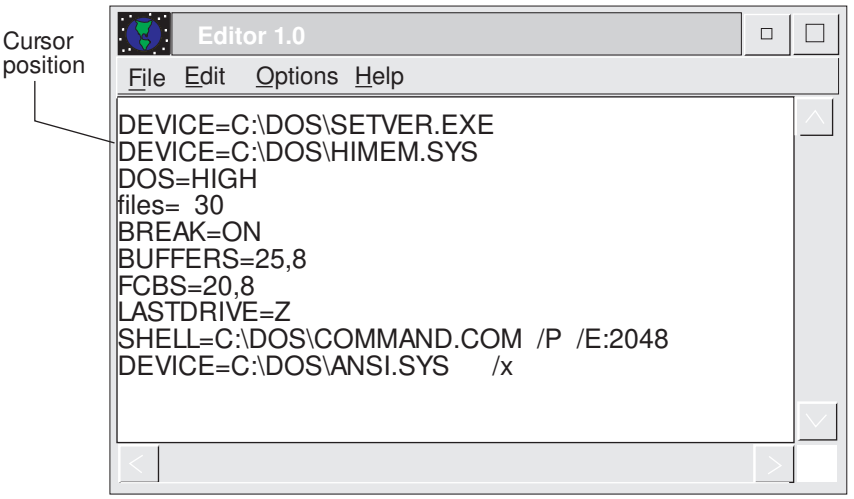


Figure 19. Example of Positioning the Cursor

Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for descriptions of these member functions.

Clipboard Operations

The `IMultiLineEdit` control has several member functions to perform clipboard operations, including `copy`, `cut`, `paste`, `clear`, and `discard`

Once you have defined an MLE and set it into a client window, use these member functions to copy text to the clipboard, cut and put text into the clipboard, or paste from the clipboard only the marked lines.

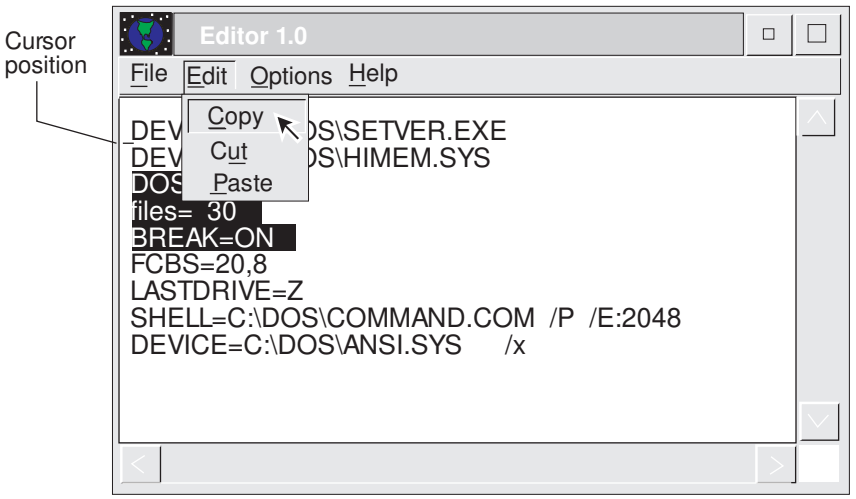


Figure 20. Example of Cutting Text to the Clipboard

Figure 20 contains marked lines and a menu option, **Edit** with a menu item, **Cut**. Suppose that the menu item ID of this menu item is MI_CUT. The following statements implement the action of cutting to the clipboard:

```
.
.
   case MI_CUT:
    mtextfield->cut();              //  cut to clipboard
    break;
.
.
```

Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for descriptions of other member functions.

Container Control

A container is a control used to hold objects. OS/2 provides a variety of containers, such as folders, templates, and the Workplace Shell itself. Containers can show their objects in different views: the tree view, the icon view, the text view, the name view, and the details view. Using the User Interface Class Library, you can develop your own containers and change views, behaviors, and layouts.

Figure 21 shows an example of a container.

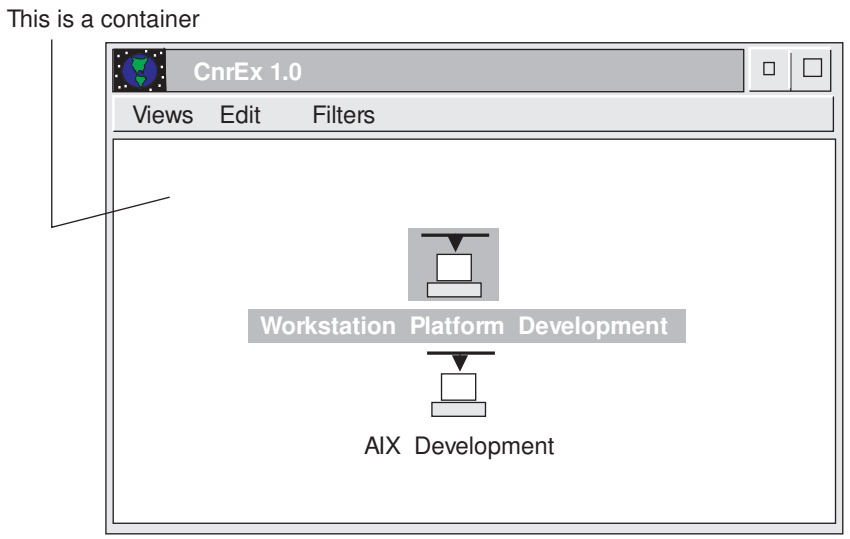


Figure 21. Example of a Container

Containers are defined by the Common User Access (CUA) architecture. For a complete description of CUA containers, refer to the *SAA CUA Guide to User Interface Design* and the *SAA CUA Advanced Interface Design Reference*.

Creating a Container

Use the `IContainerControl` class to create an instance of a container object. This class allows you to control, for example, the view of the objects inside the container. To create a container, use the following statement:

```
IContainerControl cnrCtl(CNR_RESID, this, this);
```

Several styles are available for containers that allow you 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:

```
.
cnrCtl = new IContainerControl (CNR_RESID, this, this);
cnrCtl->setMultipleSelection();
.
```

Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* to learn about other styles and related member functions.

Creating an Instance of a Container Object

A container has no meaning without its objects. Using the `IContainerObject` class, you can create the objects to be used in the container.

The following statement is an example of an `IContainerObject` constructor:

```
IContainerObject  ( const IString&        string,
                    const IPointerHandle&   iconHandle =  );
```

At a minimum, an `IContainerObject` has an Icon and a Text(Name).

Because you will usually want to design your own objects for your applications, you should create a class that is derived from the `IContainerObject` class. For example, suppose you want to create a container object with information such as names, addresses, and ZIP codes, for a department in a company. You can define this class as follows:

```
class Department : public IContainerObject
{
   public:
       Department(const IString& Name,
                  const IPointerHandle& Icon,
                  const IString& Code,
                  const IString& Address,
                  ACnrexWindow* win);

       IString Code()
       const {  return strCode; }

       IString Address()
       const {  return strAddress; }

       void setCode (IString code)
       {strCode = code;}

       void setAddress (IString address)
       {strAddress = address;}

   private:

      IString strAddress;
      IString strCode;
};
```

The statements for a constructor definition are:

```
Department :: Department(const IString& Name, const IPointerHandle& Icon,
            const IString& Code, const IString& Address, ACnrexWindow* win):
       IContainerObject(Name, Icon),
       strCode (Code),
       strAddress (Address),
       Mywin(win)
       {}
```

After the class is defined, you can create an instance of an object using the following statements:

```
dept1 = new Department (
 "Workstation Platform Development",
 IApplication::current().userResourceLibrary().loadIcon(OSLOGO),
 "TWPD",
 "Building 71",
 this);
```

Adding and Removing Objects

Once you have created the objects and the container, you need to add the objects into the container. To add an object, use the following statement:

```
cnrCtl->addObject(dept1);
cnrCtl->addObject(dept2,dept1);
cnrCtl->addObject(dept3,dept1);
cnrCtl->addObject(dept4,dept1);
cnrCtl->addObject(dept5);
```

To add objects in a certain order, use the following highlighted statement:

```
cnrCtl ->addObject(dept1);     //WorkStation Platform Development
cnrCtl->addObject(dept2,dept1); // UI Development
cnrCtl->addObject(dept3,dept1); // Platform I
cnrCtl->addObject(dept4,dept1); // Edit and Services
cnrCtl->addObject(dept5);      // AIX Development
```

where *dept2* is an object of the same class and constructor as *dept1* and, in the hierarchy view, *dept2* appears under *dept1*.

When you place the container in the client window and show the window and the container, you see a window similar to the one in Figure 22 on page 90:

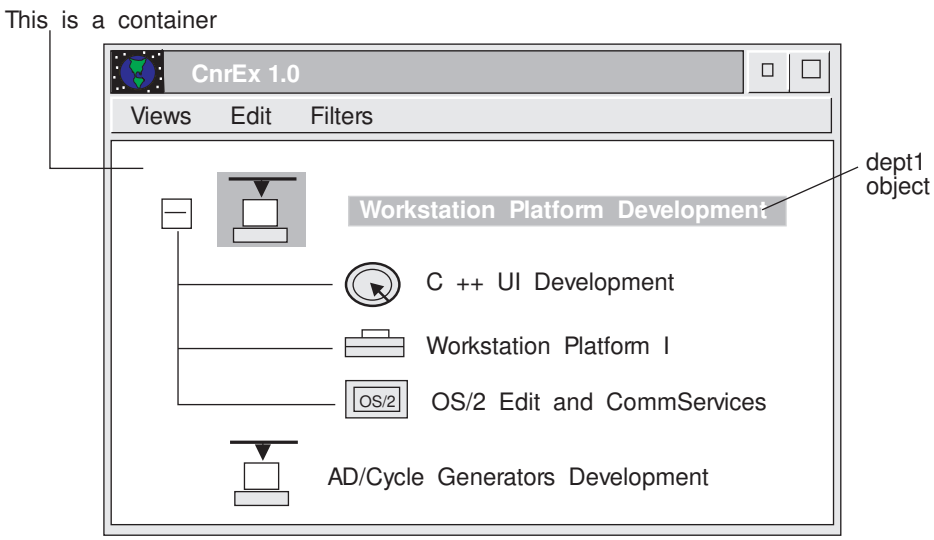


Figure 22. Example of a Container Showing a Tree View

The window in Figure 22 shows a tree Icon view of the container's objects. This view is discussed later.

Filtering Objects

The User Interface Class Library allows you to filter objects in a container. The container uses the `FilterFn` nested class to show a subset of the existing objects by filtering out some of the objects.

First, define a class derived from `FilterFn` and then override the member function `isMemberOf` . When you apply the `filter` member function of the `IContainerControl` class, the member function `isMemberOf` of the `FilterFn` class receives the container objects and the container itself, and returns true or false. If true is returned, the container object remains displayed in the container; however, if false is returned, the object is hidden.

Overriding the member function `isMemberOf` allows you to code the conditions of a valid object. The following example shows a `FilterFn` class definition:

```
class OnlySelectedObjects : public IContainerControl::FilterFn
{
virtual Boolean
```

```
  isMemberOf( IContainerObject* object,
              IContainerControl* container) const
  {
    return isSelected(object);
  }
};
```

The `isSelected` function returns true if the object has selection emphasis. Refer to *IBM C/C++ Tools: User Interface Class Library Reference* for information about the types of emphasis.

After the class is defined, you can create an instance of the `FilterFn` object and use the `filter` member function using the following statements:

```
OnlySelectedObjects onlySelectedObjects;
cnrCtl->filter(onlySelectedObjects);
```

Figure 23 on page 92 shows how the container appears before and after you apply the filter:

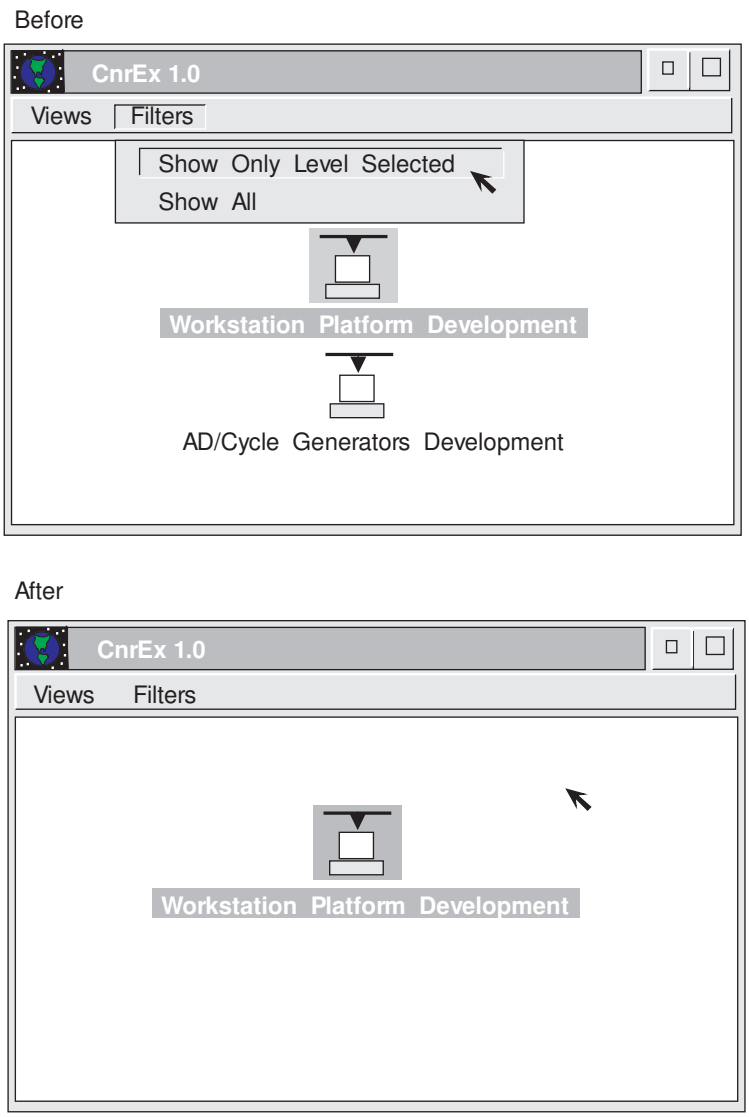


Figure 23. Example of Filtering Container Objects

Cursors and Containers

You can use an object cursor to apply an action to a group of objects or to know which objects have a specific emphasis. Use the `ObjectCursor` nested class to iterate through a collection of container objects.

The following example creates an `ObjectCursor` and uses it to set the emphasis selected to all container objects:

```
IContainerControl::ObjectCursor CO1 (cnrCtl);

for (CO1.setToFirst(); CO1.isValid(); CO1.setToNext())
  {
  cnrCtl->setSelected(cnrCtl.objectAt(CO1));
  }
```

Figure 24 on page 94 shows the result of setting the selection emphasis:

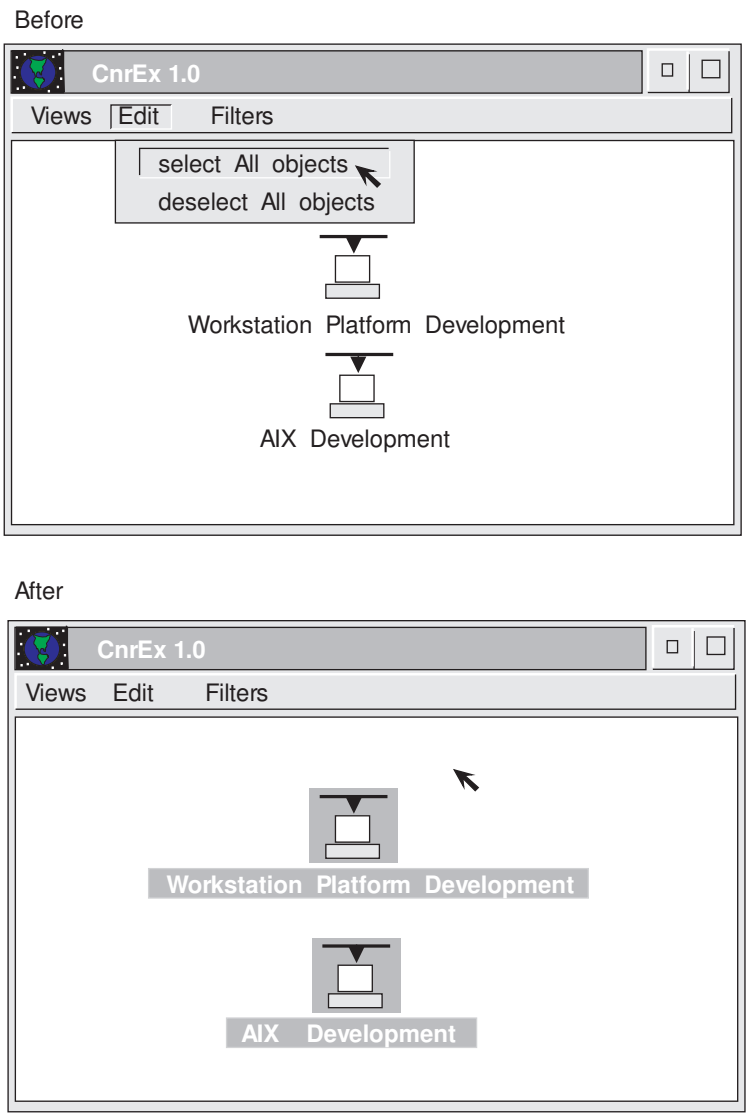


Figure 24. Example of Using an Object Cursor

Working with Views

You can use most of the views provided by the User Interface Class Library by using the corresponding member function. For example, the following statement uses the member function that causes a container to display the icon view:

```
cnrCtl->showIconView();
```

The statement above provides the container view shown in Figure 25:

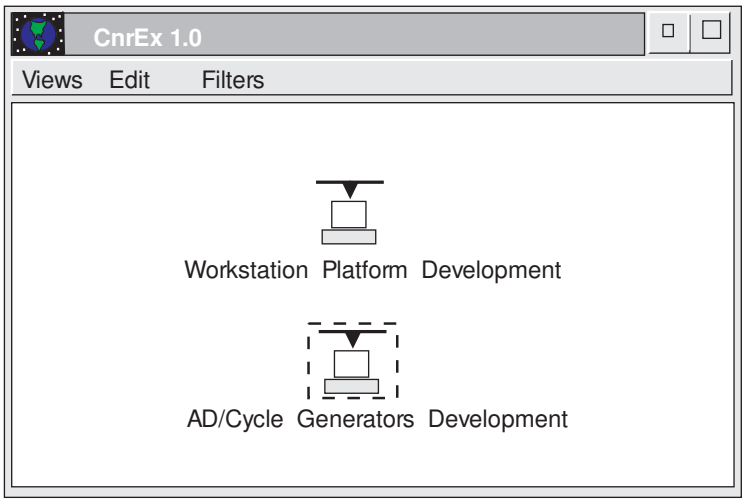


Figure 25. Example of the Icon View

The following statement provides the tree icon view:

```
cnrCtl->showTreeIconView();
```

Figure 26 on page 96 shows a container with the tree icon view:

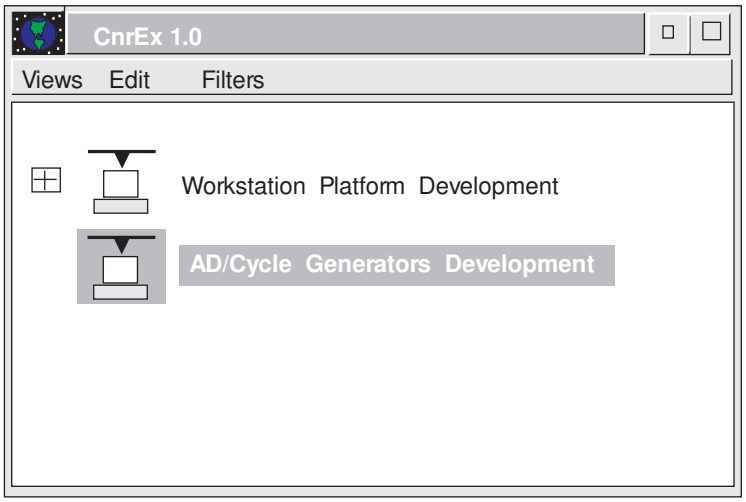


Figure 26. Example of the Tree Icon View

The "+" sign indicates the tree can be expanded, as shown in Figure 27

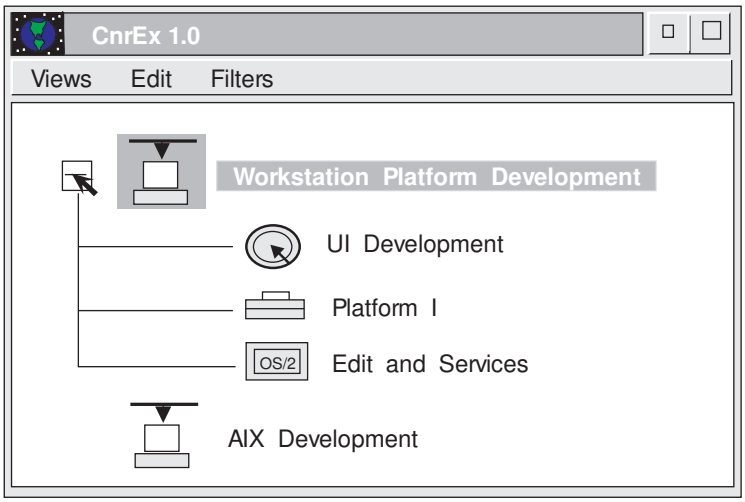


Figure 27. Example of an Expanded Tree Icon View

Container Columns and Details View

Use the `IContainerColumn` class to show a details view from a container object in a container. Use this class to set text in the heading of the columns, add horizontal and vertical separators by column, and align the column contents.

One way to create an instance of a `IContainerColumn` is 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.

The following is an example of the constructor for this class:

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

To create an instance of a container column, use the following statements:

```
colIcon = new IContainerColumn (IContainerObject :: iconOffset(),
          IContainerColumn::defaultHeadingStyle (),
          IContainerColumn::icon |
          IContainerColumn::alignVerticallyCentered);

colName = new IContainerColumn (IContainerObject::iconTextOffset(),
          IContainerColumn::defaultHeadingStyle (),
          IContainerColumn::string |
          IContainerColumn::alignVerticallyCentered |
          IContainerColumn::alignLeft |
          IContainerColumn::horizontalSeparator);

colCode = new IContainerColumn (offsetof(Department, strCode));

colAddress = new IContainerColumn (offsetof(Department, strAddress));
```

Use the `IContainerObject` member functions `iconOffset` and `iconTextOffset` with the C++ function `offsetof` to obtain the necessary offsets.

In the previous example, colIcon, colName, colCode, colAddress are defined as members of an IFrameWindow. The statements look like this:

```
private:                              //Define private information
  IContainerControl * cnrCtl;
  Department *dept1, *dept2, *dept3, *dept4, *dept5, *dept6, *dept7;
  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 ");
```

Use the member function `showSeparators` to add a vertical separator after a column or a horizontal separator under the heading text. By default, both are added. 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, you can add them into the container using the following statements:

```
cnrCtl->addColumn(colIcon);
cnrCtl->addColumn(colName);
cnrCtl->addColumn(colCode);
cnrCtl->addColumn(colAddress);
```

Figure 28 is an example of a details view of a container.

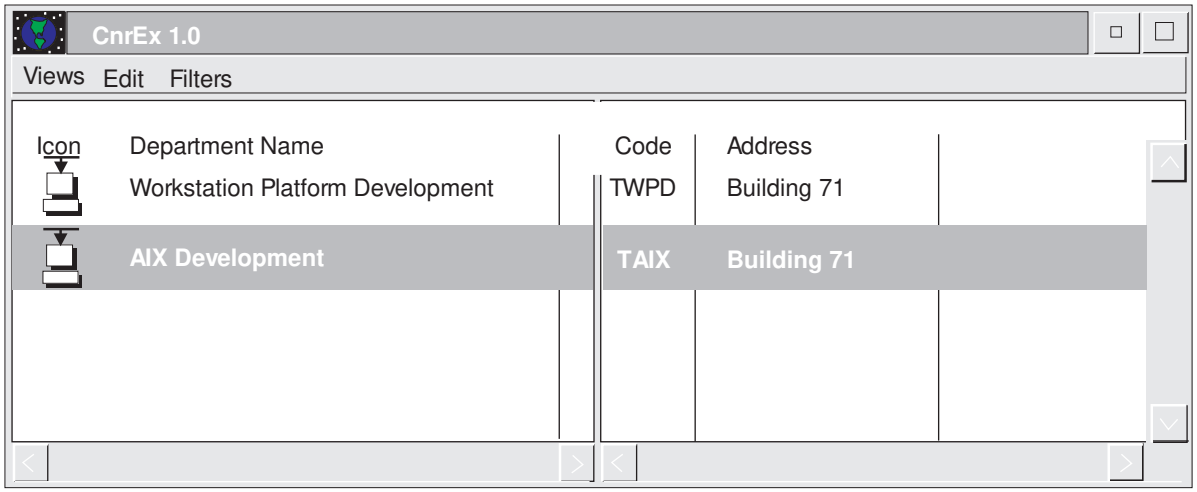


Figure 28. Example of the Details View

Use the following statements 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.

```
cnrCtl->setDetailsViewSplit(colName, 15 );
```

The following statement deletes all objects in the container when the container is deleted. By default, the container only removes objects, but does not delete them when the container is deleted.

```
cnrCtl->setDeleteObjectsOnClose();
```

Creating a Pop-up Menu in a Container

To create a pop-up menu in a container, create a subclass of `ICnrMenuHandler` and override the member function `makemenu`to handle the pop-up menu events. Use the `setCnr` member function to set the container control and make it visible for our class. The following statements create class:

```
class ACnrMenuHandler: public ICnrMenuHandler //
{
  public:
    setCnr(IContainerControl * pcnr)  { pcnrCtl = pcnr; }
  protected:                        //Define Protected Member           .
    IPopUpMenu* makePopUpMenu(const IMenuEvent& cnEvt);
  private:
    IContainerControl * pcnrCtl;
};
```

After overriding the `makePopUpMenu` member function, you can add you own statements. The following statements create a pop-up menu displayed next to a container object with source emphasis:

```
IPopUpMenu * ACnrMenuHandler :: makePopUpMenu(const IMenuEvent& cnEvt) //
{                                  //
  IPopUpMenu * popUp;                 //Define popUp variable
  if (popupMenuObject()) {
  popUp = new IPopUpMenu (ID_POPMENU,   //Create pop-up menu with AutoDelete on
            cnEvt.window());         // from a resource id.
  popUp->show(cnEvt.mousePosition());   //Show pop-up menu
  pcnrCtl->showSourceEmphasis(popupMenuObject()); //Put source emphasis on the
                                        // from where the pop-up menu
                                        // was called
  return popUp;                     //Return pop-up menu
  }
  else
  return  ;
};
```

Figure 29 on page 101 shows the pop-up menu in a container object.

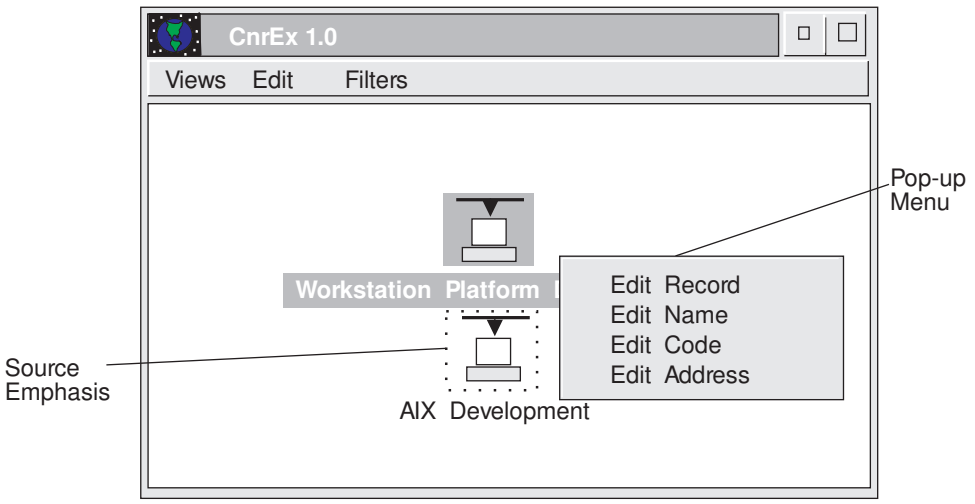


Figure 29. Example of a Pop-up Menu in a Container Object

Notebook Control

Use the `INotebook` class to create and manage the notebook control window. You can create an instance of this class in the following ways:

```
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);
```

The default style of this class is with solid binding, square corner tabs, and the status line text is left-justified. To change the style, define an instance of the `INotebook::Style` class and initialize it. For example:

```
INotebook::Style style = INotebook::spiralBinding |
                         INotebook::roundedTabs ;
```

The notebook created using the preceding statements has a spiral binding and rounded corner tabs. Figure 30 on page 102 shows an example of a notebook control.

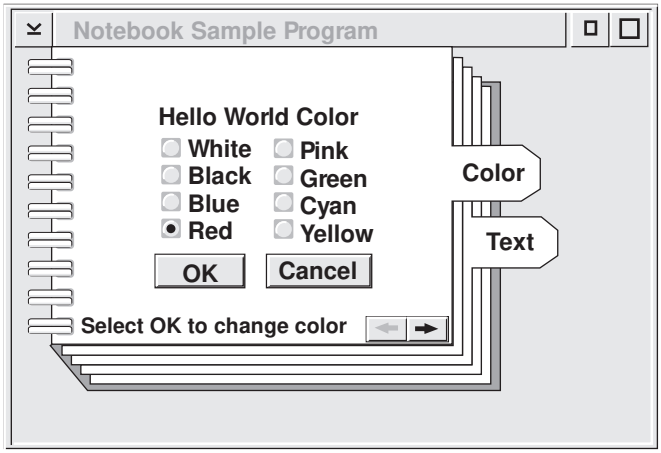


Figure 30. Notebook Control Example

Notebook Styles

Use the following notebook styles to customize you notebook controls:

- Type of binding

  `spiralBinding` Draws a spiral binding.

  `solidBinding` Draws a solid binding. This is the default.

- Intersection of back pages

  `backPagesBottomRight` Draws the back pages of the notebook behind the bottom right corner. This is the default.

  `backPagesBottomLeft` Draws the back pages of the notebook behind the bottom left corner.

  `backPagesTopRight` Draws the back pages of the notebook behind the top right corner.

  `backPagesTopLeft` Draws the back pages of the notebook behind the top left corner.

- Location of major tabs

  `majorTabsRight` Draws the major tabs on the right side. This is the default.

  `majorTabsLeft` Draws the major tabs on the left side.

  `majorTabsTop` Draws the major tabs on the top side.

  `majorTabsBottom` Draws the major tabs on the bottom side.

- Shape of tabs

  `squareTabs` Draws tabs with square corners. This is the default.

  `roundedTabs` Draws tabs with rounded corners.

  `polygonTabs` Draws tabs with polygon-shaped corners.

- Alignment of status line text

  `statusTextLeft` Left justifies the status line text.

  `statusTextRight` Right justifies the status line text.

  `statusTextCenter` Centers the status line text. This is the default.

- Alignment of text associated with tabs

  `tabTextLeft` Left justifies the text in the tabs. This is the default.

  `tabTextRight` Right justifies the text in the tabs.

  `tabTextCenter` Centers the text in the tabs.

The User Interface Class Library also provides functions to change the size of a notebook's parts:

`setMajorTabSize` Sets the size of the major tabs (in pixels).

`setMinorTabSize` Sets the size of the minor tabs (in pixels).

`setPageButtonSize` Sets the size of the arrow buttons used to turn the notebook's pages (in pixels).

You can create a notebook, specify its style, and change the size of major tabs and minor tabs with the following statements:

```
INotebook    *pnoteBook;

pnoteBook= new INotebook ( ID_NOTEBOOK, this , this,
                      IRectangle(),
                      INotebook::spiralBinding |
                      INotebook::backPagesTopRight |
                      INotebook::majorTabsRight |
                      INotebook::statusTextLeft |
                      IWindow::visible);

pnoteBook->setMajorTabSize(ISize(6 ,3 ));
pnoteBook->setMinorTabSize(ISize(8 ,4 ));
```

Version 6 of the Hello World example creates a notebook using these statements in ACOLORW6.CPP:

```
INotebook       * notebook;

notebook=new INotebook(WND_COLOR_NOTE, this, this, IRectangle(),
                   INotebook::defaultStyle());
```

Page Settings

Use the `INotebook` class in conjunction with the `PageSettings` class. The page settings objects allow the user to change and set information about pages in a notebook.

The following example of `PageSettings` is taken from the file ACOLORW6.CPP used in Version 6 of the Hello World example:

```
IMultiCellCanvas * staticCanvas;
INotebook::PageSettings    staticPage;

staticCanvas=new IMultiCellCanvas( WND_STATIC_COLOR, notebook, this,
                             IRectangle());
INotebook::PageSettings::Attribute            //Define the page attributes
  attribute=INotebook::PageSettings::majorTab    //  with a major tab and
   | INotebook::PageSettings::autoPageSize;      //  AutoPageSize
staticPage=INotebook::PageSettings(attribute);   //Create static color page
staticPage.setTabText("1");                  //Set tab text
notebook->addFirstPage(staticPage,             //Add as first page to notebook
staticCanvas);                              //
```

This page is created on a multi-cell canvas. The tab text is set to 1 and this page is inserted into the notebook as the first page.

Chapter 7. Advanced Topics

Extending the Event Handling

The User Interface Class Library provides handlers for most of the common PM messages. However, you may find it necessary to process messages for which there are no predefined handler classes. The User Interface Class Library makes it easy to add new event and handler classes seamlessly.

The `IHandler` class is designed to act as a base class for handlers. All event handlers should be derived from this class. The following steps are required to provide a handler:

1. Subclass the `IHandler` class.
2. Override the `dispatchHandlerEvent` member function.
3. Provide a virtual function (callback) to process the message.

The following example shows how to provide a handler for PM timer events.

1. Class declaration for `ATimerHandler`.

   The class is derived from `IHandler` and provides a virtual function `timer` to process the event.

```
class ATimerHandler : public IHandler
{
public:
  /* use default constructor */

  Boolean
    dispatchHandlerEvent( IEvent& evt );

protected:
virtual Boolean
   timer( IEvent& evt );
};
```

© Copyright IBM Corp. 1993

2. Override `dispatchHandlerEvent` member function.

   The function must determine the relevance of the message. If the message is not relevant, the function returns false and passes the message to other handlers attached to the window.

```
Boolean ATimerHandler::dispatchHandlerEvent( IEvent& evt )
{
  if ( evt.eventId() == WM_TIMER )    //If timer event call
     return timer( evt );           // function to process
                                 //note: WM_TIMER is defined in the
  return false;                    // OS/2 Developer's Toolkit
}
```

3. The `timer` member function.

   This provides a default return. This class acts as a base class. The default provides safe behavior when you create an instance of the class.

```
Boolean ATimerHandler::timer( IEvent& evt )
{
  return false;
}
```

The `ATimerHandler` class encapsulates the WM_TIMER messages generated by PM. You can derive a class from `ATimerHandler` and override the `timer` member function to provide whatever event handling is required.

To prevent users of this class from having to understand how information is encoded in the two message parameters inside the event, you should 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 ) {;}  // Define functions inline

  unsigned long
    timerId() const { return parameter1().number1(); }
};
```

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

Change the `dispatchHandlerEvent` to create an instance of `ATimerEvent`. You should also change the `timer` to accept an `ATimerEvent` object as a parameter.

```
Boolean ATimerHandler::dispatchHandlerEvent( IEvent& evt )
{
  if ( evt.eventId() == WM_TIMER )    //If timer event call
     {                            // function to process
     ATimerEvent timerEvt( evt );
     Boolean rc = timer( timerEvt );  //Call timer to process
     evt.setResult(timerEvt.result()); //Move results to event
     return rc;                   //Return with return code
     }

  return false;
}
```

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

When adding handlers, it is often useful to restrict the window classes to which the handler can be attached. A handler class can override the `handleEventsFor` and `stopHandlingEventsFor` member functions to provide a certain degree of type safety.

The following example shows how to restrict the class of windows to which the timer class can be attached to the `ITextControl` class and its derived classes:

1. The class declaration.

```
class ATimerHandler : public IHandler
{
public:
  /* use default constructor */

Boolean
   dispatchHandlerEvent( IEvent& evt );

virtual ATimerHandler
   &handleEventsFor      ( ITextControl* textWindow ),
   &stopHandlingEventsFor ( ITextControl* textWindow );
```

```
protected:
virtual Boolean
   timer( ATimerEvent& evt );

private:                              //Make these functions private
virtual IHandler                       // so they cannot be called
   &handleEventsFor      ( IWindow* window ),
   &stopHandlingEventsFor ( IWindow* window );
};
```

2. Override `handleEventsFor` to accept only `ITextControl` objects.

```
ATimerHandler &ATimerHandler::handleEventsFor( ITextControl* textWindow )
{
  IHandler::handleEventsFor( textWindow );  //Call parent class
  return *this;                          // member function
}
```

3. Override `stopHandlingEventsFor` to accept only `ITextControl` objects.

```
ATimerHandler &ATimerHandler::stopHandlingEventsFor( ITextControl* textWindow )
{
  IHandler::stopHandlingEventsFor( textWindow ); //Call parent class
  return *this;                               // member function
}
```

Tracing

Use the trace class, `ITrace`, together with related macros to simplify the process of adding tracing code to an application. Using the trace functions, you can write trace output to STDOUT (standard output stream), STDERR (standard error stream), or an OS/2 queue. You can control the trace options using environment variables or by statements in your program.

The environment variables, ICLUI TRACE and ICLUI TRACETO, provide the default tracing options.

ICLUI TRACE has three valid values:

OFF Set trace off. This is the default.

ON Set trace on.

NOPREFIX Set trace on, but no prefix information is written to trace.

ICLUI TRACETO has three valid values:

QUEUE Trace is written to a 32-bit named OS/2 queue. The name is \\QUEUES\PRINTF32. This is the default.

STDOUT Trace is written to the standard output stream.

STDERR Trace is written to the standard error stream.

The following example shows how to write trace information:

```
#include <itrace.hpp>                        //Include trace class
/*  ... function to trace  ... */
void myFunction(int x)
{
  ITrace trc("myFunction");                  //Create an ITrace object
  trc.write("now at this point");             //Use static member function
  ITrace::write(IString("the value is = ") + IString(x));  //      write
  return;
}
```

If you provide message text, the `ITrace` instance writes a message during its constructor and destructor, thus indicating the start and end of the function. Because of the performance overhead of tracing, you may want to limit your use of the trace code to the development and test phases. For example, to run the preceding example program after testing it, you remove the tracing lines and recompile the program.

A more flexible approach to tracing is to use the predefined User Interface Class Library macros. These macros expand to calls to the trace function only if another macro is defined. Using this approach, the example becomes:

```
#define  IC_TRACE_DEVELOP                    //Define trace level
#include <itrace.hpp>                        //Include trace class
/*  ... function to trace  ... */
void myFunction(int x)
{
  IFUNCTRACE_DEVELOP();                      //Trace entry and exit
  ITRACE_DEVELOP("now at this point");
  ITRACE_DEVELOP(IString("the value is = ") + IString(x));
  return;
}
```

For PM programs, information written to the standard output stream and standard error stream is discarded. If you start the program from the command line, you can redirect these streams to a file or named pipe. The commands to redirect the stream to a file are as follows:

```
[C:\]hello1 >stdout.lst      <- redirect stdout to file stdout.lst
[C:\]hello1 2>stderr.lst     <- redirect stderr to file stderr.lst
```

An example of the trace output is shown below:

```
<--- prefix -----> <----- trace ----------------->

        9    595: 1 +myFunction(int x)(121)           <- function entry
       1     595: 1   >now at this point
       11    595: 1   >the value is = 5
       12    595: 1 -myFunction(int x)               <- function exit
```

The prefix area shows the trace line number, the process ID, and the thread ID. The IFUNCTRACE_DEVELOP macro automatically generates the trace lines that show the entry and exit from the function. The number in brackets after the parameter list is the source code line number of the macro. The I_TRACE_DEVELOP macro produces the other two lines.

If the IC_TRACE_DEVELOP macro is defined, the trace statements are generated; otherwise, no trace statements are generated. This means that after testing is complete, it is not necessary to remove all the trace lines. Remove the macro and recompile the code as usual.

Exception Handling

The User Interface Class Library uses the C++ exception handling mechanism to return errors to the application. Several different classes of exception objects can be thrown. Because all these classes are derived from the `IException` class, an application can catch specific exceptions or all exceptions.

The following table lists the exception classes and the situations in which they are typically thrown.

Figure 31. Exception Classes

Exception Class	Description
`IAccessError`	Thrown when a logical error occurs, such as "resource not found"
`IAssertionFailure`	Thrown when the expression in an IASSERT macro evaluates to false
`IDeviceError`	Thrown when a hardware related error occurs
`IInvalidParameter`	Thrown when an invalid parameter is passed
`IInvalidRequest`	Thrown when an object is in the wrong state for a function
`IResourceExhausted`	Thrown when a resource is exhausted or currently unavailable
`IOutOfMemory`	Thrown when heap storage is exhausted
`IOutOfSystemResource`	Thrown when an OS/2 resource is exhausted
`IOutOfWindowResource`	Thrown when a PM resource is exhausted

Typically, an application surrounds a function that might fail with a try-catch block. The following example shows how an application attempts to set its default resource library. If this fails, an `IAccessError` is thrown and the example code explicitly handles the exception. The application passes on any other exception that is thrown:

```
try
   {                                  //Try to use notfound.dll
   IApplication::current().setUserResourceLibrary("NOTFOUND");
   }
catch (IAccessError &exc)                  //Catch only access errors
   {                                  //DLL probably not in libpath
   const char *exText = exc.text();
   unsigned long exId = exc.errorId();
   /* ... add code to process the exception ... */
   }
```

Each exception object thrown contains an error number, a severity indicator, one or more lines of text, and information about where the exception was thrown. The `IException` class provides accessor functions to extract this information from the object. The member function `textCount` retrieves the number of lines of exception text, and the member function `text.` retrieves the exception text.

The `ITHROW`, `IASSERTSTATE`, `IASSERTPARAM`, and `ITHROWGUIERROR` macros throw all exceptions in the library and the `RETHROW` macro rethrows the exceptions. These macros automatically insert into the exception object the line and program file in which the exception was thrown. These macros also log the exception information. By default, exception information is written to the same destination as the trace output. However, you can provide your own function by deriving a class from `IException::TraceFn`, overriding the `write` virtual function, and registering it using `IException::setTraceFunction`.

Note: C++ exceptions are not the same as OS/2 exceptions.

Providing a Default Exception Handler

The C++ exception mechanism passes exceptions back up the function call chain until a try-catch block is found that handles the exception. Because most processing in a User Interface Class Library application is a result of events, this usually results in the uncaught exceptions being passed to PM which, in turn, causes the application to terminate in an unpredictable way.

The User Interface Class Library allows an application to register a default exception handler. The event dispatching loop catches any exception thrown in a handler or function called from a handler, and passes it to the registered default exception function. This allows the application to either try to continue or to terminate in a controlled way.

The steps to register a default exception handler are as follows:

1. Subclass the `IWindow::ExceptionFn` class.
2. Override the `handleException` member function.
3. Create an instance of this class.
4. Register using `IWindow::setExceptionFunction`

The following example shows how to create a default handler that uses the tracing functions to log the exception and displays the information in a message box.

1. Subclass `IWindow::ExceptionFn`.

   The class declaration.

   The class has a single constructor that requires a frame window in which the handler displays a message box. The frame window acts as the owner of the message box.

```
class AExceptionFn : public IWindow::ExceptionFn
{
public:
   AExceptionFn(IFrameWindow *frame) : owner(frame) {;}
Boolean
   handleException (IException& exception, IEvent& event);
private:
   IFrameWindow *owner;
};
```

2. Override `handleException`.

   The member function definition.

   The last of the text messages of the exception object is written to the trace output and displayed in a message box. The function returns true to indicate that the exception should not be rethrown.

```
Boolean AExceptionFn::handleException (IException& exception, IEvent& event)
{
  IFUNCTRACE_DEVELOP();                       //Trace function entry/exit
  unsigned long cnt  = exception.textCount();
  const char *text = (cnt >  ) ? exception.text( cnt-1 )
                        : "No error text available" ;
  IString str( text );
  ITRACE_DEVELOP( exception.name() );
  ITRACE_DEVELOP( IString("text count = ") + IString(cnt) );
  ITRACE_DEVELOP( str );
  IMessageBox msgbox( owner );                 //Create message box
  msgbox.setTitle( exception.name() );
  msgbox.show( (char *)str ,
             IMessageBox::okButton       |
             IMessageBox::informationIcon  |
             IMessageBox::applicationModal |
             IMessageBox::moveable       );
  return true;                              //Stop rethrow of exception
}
```

3. Create an object of this class.

   The class definition of the frame window.

   The object is part of our main application window object.

```
class aListBoxWindow : public IFrameWindow
{
  public:
    aListBoxWindow(unsigned long windowId);    //Constructor for this class
    /* ... other public member functions ... */
  private:
    AExceptionFn   excptHandler;
    /* ... other private data ... */
};
```

4. Register using `IWindow::setExceptionFunction`.

   The exception function is created in the constructor for the window and then registered.

```
aListBoxWindow::aListBoxWindow(unsigned long windowId)
  : IFrameWindow( IFrameWindow::defaultStyle() |
                  IFrameWindow::minimizedIcon,
                  windowId) ,
    excptHandler( this )
{
  setExceptionFunction(&excptHandler);
  /* ... rest of constructor code ... */
}
```

Threads and Protecting Data

The User Interface Class Library provides classes to implement multi-threaded programs. The primary class used to deal with threads is `IThread`. Instances of this class represent separate threads of execution and provide the ability to start and stop the thread, set various thread attributes, and determine the default environment for the thread. In addition, the `ICurrentThread` class allows you to set and query attributes for the currently executing thread, start event processing, and suspend the current thread until another thread has terminated.

Current Thread

There is only a single instance of the class for each thread, and it can be accessed using the following statement:

```
ICurrentThread  curThread = IThread::current();
```

This static data member allows access to some information held on a per-thread basis. The member also allows access to some functions that can be applied only to the current thread. One example is the initialization of the PM environment for a thread. A thread without a PM environment can initialize one and later terminate it using the following statements:

```
IThread::current().initializePM();
/* ... do thread processing ... */
IThread::current().terminatePM();
```

If necessary, the thread can enter its event processing loop using:

```
IThread::current().processMsgs();
```

Starting a Thread

You can start a thread of execution using the `IThread` class. Once started, the instance of `IThread` provides a means of querying and stopping the thread. The thread and the instance of `IThread` are independent; therefore, when the instance of IThread is destroyed, the thread is unaffected.

The function to be dispatched on a separate thread can be either a member function or a non-member function. If you create an instance of `IThread` with the function, a thread is created and dispatched immediately. Alternatively, you can create an instance of the class and later dispatch it. This has the advantage of allowing you to set parameters that affect the execution of the thread prior to dispatching.

Starting Non-member Functions

The `IThread`class dispatches non-member functions with either of the following two function prototypes:

```
void (_Optlink *)(void *)
void (_System *)(unsigned long)
```

This provides support for migrating code that uses either `_beginthread` or `DosCreateThread` to start the function. The linkage directives, `_Optlink` and `_System`, are discussed in the *C++ Compiler Reference*. The linkage directive, `_Optlink`, is the default. The following examples assume this default.

To start a thread with the default environment and default options, the following statements are needed:

```
void threadFn(void *pvParms);     //Function to run on separate thread
void    *pv;

IThread  thread(threadFn, pv);    //Dispatch thread with default environment
```

The following example shows how to set some of the options before dispatching the thread. The environment is created before the function is called, and appropriate cleanup action are taken after it terminates:

```
IThread  thread;                //Assume PM environment
thread.setStackSize(65536)        //Set 64K stack size
thread.setQueueSize(32)          //32 elements in PM queue
thread.start(threadFn, pv);       //Dispatch thread
```

Other functions also exist to change the priority level of the thread, although for threads that process events, changing the priority can adversely affect the overall performance of the system.

Once you have started a thread, you can suspend, resume, or stop the thread. You can also query its thread ID. The following example stops the thread if it has a thread ID of 2:

```
void    *pv;
IThread  thread(threadFn, pv);   //Dispatch thread with default environment
/* ... let thread process ... */
if (thread.id() == IThreadId(2))   //If thread ID is 2, then stop it
  thread.stop();
```

Because threads often require that a PM environment has been established before they can do their work, the User Interface Class Library automatically establishes a PM environment for all threads created in a PM application. If this is not necessary, a thread can request that this initialization be skipped. For example:

```
void threadFn(void *pvParms);         //Function to run on separate thread
void    *pv;

IThread  thread(threadFn, pv, false);  //No PM environment
```

Starting a Member Function

Use the `IThread` class to start member functions. Direct support is provided for starting member functions that have no parameters, but you can also start functions that have parameters.

To start a member function that takes no parameters, use the following statements:

1. Create an instance of the template class `IThreadMemberFn`.
2. Start a thread and pass the instance as a parameter.

The following example shows how to execute the function `AClass::longFn` on a separate thread:

Create an instance of the template class with the class that contains the member function. Create the instance of the template class with the `new` operator so that the instance can be destroyed automatically when the thread ends. The two parameters on the constructor are the object for which the member function is called and the member function itself, as shown in the following example:

```
/* function to run is ...  void AClass::longFn()    */
AClass object;                 //Object to run member function against

IThreadMemberFn<AClass> *aMemberFn =
                 new IThreadMemberFn<AClass>( object
                                       , AClass::longFn );
IThread  thread( aMemberFn );     //Dispatch thread
```

To start a member function that takes parameters, use the following statements:

1. Derive a class from the `IThreadFn` class.
2. Define constructor that takes an object of the class and the parameters you want to pass
3. Override the `run` member function to call the member function.
4. Create an instance of the derived class.
5. Start a thread and pass the instance as a parameter.

The following example shows how to start a function: `AClass::longFn(int, IString).`

1. The class declaration.

   The class is derived from the `IThreadFn` class. It has a single constructor that requires an instance of the `AClass` class and the two parameters. The class overrides the virtual function `run` and calls the required member function, as in the following example:

```
class AClass
{
  public:
    void longFn(int, IString);
  /* ... rest of class declaration ... */
};

                    //This class runs the member function
                    // AClass::longFn(...) on a separate thread
class AThreadLongFn : public IThreadFn
{
  public:
    AThreadLongFn(AClass &obj, int i, IString str)
      : object( obj )
      , value( i )
      , string( str ) {;}
    void run() { object.longFn( value, string ); }
  private:
    AClass   &object;
    int      value;
    IString   string;
};
```

2. Create an instance and dispatch.

   As before, create the instance using the `new` operator so that it can be destroyed automatically:

```
AClass  object;                 //Object to run member function against
int    number = 6;
IString greeting( "Hello" );

/* function to run is ...  void AClass::longFn(int, IString)    */
                           //Create object
AThreadLongFn  *aMemberFn = new AThreadLongFn( object, number, greeting );
IThread    thread( aMemberFn );    //Dispatch thread
```

Protecting Data

If your applications have multiple threads, you typically need to serialize access to certain resources. The User Interface Class Library provides several classes to assist you. Use the `IPrivateResource` class to serialize access to a resource within a single process. The `ISharedResource` class extends this ability by providing a lock that can also be used between processes.

The simplest way to serialize access to a function is to provide a static instance of the `IPrivateResource` class. You can use this instance in association with the `IResourceLock` class to control access.

In the following example, the function guarantees that only one thread accesses it at one time:

```
static  IPrivateResource  resourceKey;    //Key must exist when function
                                          // called
void serializedFunction()
{
IResourceLock resLock(resourceKey);       //Create lock
/* ... serialized code ... */
}                                         //Lock freed with resLock destructed
```

When a thread calls `serialzedFunction`, it is blocked until any other thread executing the function exits it. This may lead to deadlock problems, so a slightly safer approach is to give a timeout value, which is the number of milliseconds that a thread can be blocked. If this time limit is exceeded an `IResourceExhausted` exception is thrown, which can then be caught.

The definition of the function becomes:

```
static  IPrivateResource resourceKey;

void serializedFunction()
{
IResourceLock resLock(resourceKey, 1  );    // timeout period =  .1 s
/* ... serialized code ... */
}
```

The code to call the function is:

```
try
   {
   serializedFunction();
   }
catch (IResourceExhausted exc)
   {
   /* ... handle failure to run function ... */
   }
```

Critical Sections

A critical section of code is a portion of code that must be executed by one thread while all other threads in the process are suspended. An example situation would be the need for one thread to modify global data while preventing other threads from accessing the data until the modifications are complete.

The User Interface Class Library provides a critical section object to handle such situations. A thread should create the critical section object before it enters a critical section and destroy when it exits the section.

The simplest way to do this is to enclose the critical section in its own block and define the object at the start of the block, as in the following:

```
{
ICritSec  lock;
/* ... do critical section processing here ... */
}                           // lock destructed when block ends
```

Because critical sections freeze the other threads in the process, you should use them with care. In addition, you should be careful when calling certain OS/2 functions within a critical section because the results may be unpredictable.

Chapter 8. Finishing Touches

Standard Dialogs

The User Interface Class Library provides a standard file dialog and a standard font dialog.

File Dialog

The `IFileDialog` class allows you to define the standard dialog for files from OS/2. Figure 32 shows an example of a file dialog:

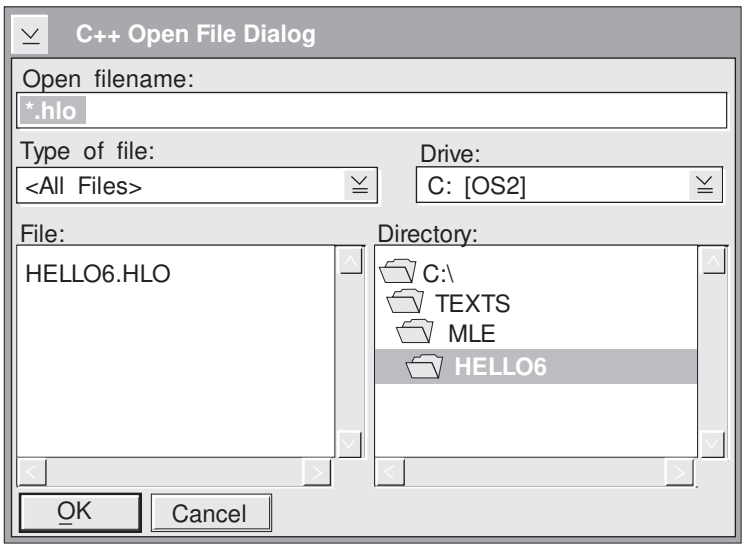


Figure 32. Example of a File Dialog

To use a dialog, follow these steps:

1. Set up the file dialog.
2. Create the file dialog.
3. Test the response.
4. Read the result.

© Copyright IBM Corp. 1993

Setting Up a File Dialog

Use this optional feature of the `IFileDialog` class to specify initial settings for the dialog you create. To use this feature, create an instance of the `Settings` class when you create the dialog, as shown in the following:

```
IFileDialog::Settings fsettings;
```

The `Settings` class has several member functions, including:

- `setOpenDialog`
- `setSaveAsDialog`
- `setFileName`
- `setPosition`

Note: Do not use the `setOpenDialog` and `setSaveAsDialog` settings together. Because these member functions perform conflicting tasks, using them together produces unpredictable results.

To set up the dialog, use the following statements:

```
fsettings.setTitle(STR_FILEDLGT);    //Set open dialog title from resource
fsettings.setFileName("*.hlo");      //Set FileNames to *.hlo
```

Creating an Instance of IFileDialog

After setting up the dialog, create an instance of the `IFileDialog` class using the following statements:

```
.
  IFileDialog * fd=new IFileDialog(  // Create file open dialog
   desktopWindow(),                   // Parent is desktop
   this,                             // Owner is me
   fsettings);                        //  with settings
```

Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for other ways to define an instance of the `IFileDialog` class.

Testing the Response from the Dialog

Use the following member function to test the response from the dialog:

`pressedOK` Returns true if the user ended the dialog by pressing OK.

Reading the Result

The result from the file dialog should be the name of a file, as illustrated in the following statement:

`fileName` Returns the fully qualified name selected by the user.

For the complete sample code, see the `openFile` member function in the AHELLOW6.CPP file (Version 6 of Hello World application).

Font Dialog

Use the `IFontDialog` class to handle fonts in your applications. Figure 33 shows an example of a font dialog.

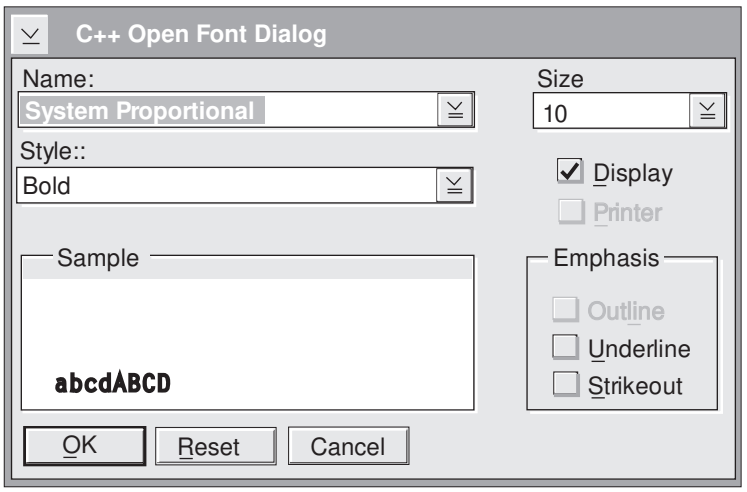


Figure 33. Example of a Font Dialog

Setting Up a Font Dialog

To set up a font dialog, parse a font instance using the `IFont` class when creating the settings:

```
IFont curfont(hello);
IFontDialog::Settings fsettings(&curfont);
```

In the preceding example, the `curfont` member function initializes the settings with the window's current font. When the user makes a selection in the font dialog, the settings font changes.

Test the Response from the Dialog

Use the following member function to test the response from the dialog:

`pressedOK` Returns true if the user ended the dialog by pressing OK.

Reading the Result

The result from the font dialog should be the font name and other characteristics such as weight and width. The following member functions return these values:

`fontFamily` Returns the font's family name.

`fontWeight` Returns the weight class (boldness) of the font.

`fontWidth` Returns the width class of the font.

Refer to "Fonts" on page 77 to see how to set up a font.

Refer to the `openFont` member function in the AHELLOW6.CPP file for complete sample code.

Message Box

The User Interface Class Library provides an `IMessageBox` class for displaying messages in a message box.

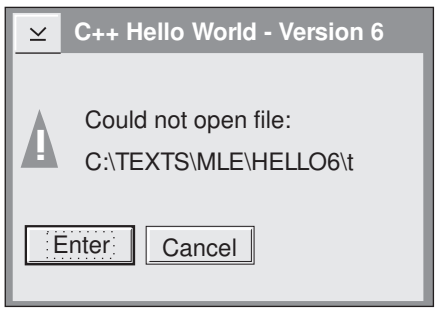


Figure 34. Example of a message box

The only way to construct instances of this class is by using an instance of `IWindow`. The instance of `IWindow` becomes the owner of the new message box. Following is an example:

```
IMessageBox  mbox(owner);
```

Some of the member functions available are:

`setTitle` Sets the title of a message box.

`show` Given a message text string, shows the message box.

The following statements show how to create a message box:

```
.
 IMessageBox msgbox(this);               //Creates an instance of IMessageBox
 msgbox.setTitle( IResourceId(STR_MSGBOX) ); //Load an String using its resource id res
 msgbox.show("This is a message", IMessageBox::okButton       |
                  IMessageBox::informationIcon  |
                  IMessageBox::applicationModal |
                  IMessageBox::moveable       );
.
```

The `displayButtonStatus` function in the AMCELCV.HPP file provides more examples of these statements.

Pop-Up Menus

A pop-up menu is a menu that is displayed next to the object with which it is associated when a user presses the appropriate key or mouse button. A pop-up menu contains choices that can be applied to an object at the time the menu is displayed.

The User Interface Class Library provides the `IPopUpMenu` class, which is inherited from the `IMenu` class, to manipulate pop-up menus. Use the `makePopUpMenu` member function to construct a pop-up menu.

There is only one way to construct objects of this class:

```
IPopUpMenu(const IResourceId& menuResId, const IWindow* owner,
         Boolean autoDelete = true);
```

where `menuResId` is a resource ID specified in the resource file. The `owner` is the object to which the pop-up menu applies. If the `autoDelete` is true, the pop-up menu object is deleted by `IMenuHandler` when it is no longer visible. The pop-up menu will not be visible until its `show` member function is called. Normally, applications will override the the `makePopUpMenu` member function in the `IMenuHandler` class and create a pop-up menu.

Version 6 of the Hello World application creates a pop-up menu object to apply to the “Hello, World” static text control area. The contents of the pop-up menu are defined in the AHELLOWE.RC resource file, as follows:

```
MENU WND_POPUP
  BEGIN
    MENUITEM "Left",   MI_LEFT
    MENUITEM "Center", MI_CENTER
    MENUITEM "Right",  MI_RIGHT
  END
```

In the AHELLOW6.HPP file, an `AMenuHandler` class is defined to create the pop-up menu.

```
class AMenuHandler: public IMenuHandler
{
  protected:
    IPopUpMenu * makePopUpMenu(const IMenuEvent& mnEvt);
};
```

The `makePopUpMenu` member function creates an `IPopUpMenu` object with the default `AutoDelete` attribute. The pop-up menu is not visible until its `show` member function is called. In this example, the WND_POPUP menu resource ID is used to create the pop-up menu.

```
IPopUpMenu * AMenuHandler :: makePopUpMenu(const IMenuEvent& mnEvt)
{
  IPopUpMenu * popUp;
  popUp=new IPopUpMenu(WND_POPUP, mnEvt.window());
  popUp->show(mnEvt.mousePosition());
  return popUp;
}
```

The AMenuHandler is created in the setupClient member function. The menu handler is set for the hello static text control.

```
Boolean AHelloWindow :: setupClient()
{
  clientWindow=new ISplitCanvas( WND_CANVAS, this, this);
  setClient(clientWindow);
  helloCanvas=new ISplitCanvas( WND_HCANVAS, clientWindow, clientWindow);
  helloCanvas->setOrientation( ISplitCanvas::horizontalSplit);
  hello = new IStaticText(WND_HELLO, helloCanvas, helloCanvas);
  AMenuHandler * mh=new AMenuHandler();
  mh->handleEventsFor(hello);
  ICommandHandler::handleEventsFor(hello);
  ..................
}
```

The selected menu item in the pop-up menu is processed by the command member function, which is used to handle command events for the frame window.

Using Help

Help information is the information about how to use a product. By describing a product's choices, objects, and interaction techniques, help information can assist users in learning to use a product.

The User Interface Class Library provides an IHelpWindow class that uses the OS/2 Information Presentation Facility (IPF) to provide help information for applications. An IHelpWindow is created and associated with one of the application's main windows. The User Interface Class Library also provides an IHelpHandler class to deal with help window events. When an application window is associated with a help window, a help event is dispatched to the handlers attached to the application window.

To create a help menu in your application window, define the `Help` submenu and the title of the help window in your resource file first. In Version 5 of the Hello World application, the help menu is defined as follows:

```
STRINGTABLE
  BEGIN
    STR_HTITLE, "C++ Hello World - Help Window"    //Help title
  END
MENU WND_MAIN
  BEGIN
    SUBMENU "~Help", MI_HELP                    //Help submenu
      BEGIN
        MENUITEM "~General help...",   MI_GENERAL_HELP
        MENUITEM "~Extended help...",  SC_HELPEXTENDED, MIS_SYSCOMMAND
        MENUITEM "~Keys help...",     SC_HELPKEYS, MIS_SYSCOMMAND
        MENUITEM "Help ~index...",    SC_HELPINDEX, MIS_SYSCOMMAND
      END
  END
```

MI_HELP is the help menu ID. The contents of the help information are stored in an IPF file, AHELLOW5.IPF. You can define a help table in the resource file to establish the relationship between the menu item ID and the panel ID that is defined in the IPF file.

```
HELPTABLE HELP_TABLE
  BEGIN
    HELPITEM WND_MAIN,       SUBTABLE_MAIN,   1
    HELPITEM WND_TEXTDIALOG,  SUBTABLE_DIALOG, 2
  END

HELPSUBTABLE SUBTABLE_MAIN                  //Main window help subtable
  BEGIN                                   //
    HELPSUBITEM WND_HELLO, 1                 //Hello <-> help ID 1
    HELPSUBITEM WND_LISTBOX,1 2             //List box help
    HELPSUBITEM MI_EDIT, 11                 //Edit menu
    HELPSUBITEM MI_ALIGNMENT, 111            //Alignment menu
    HELPSUBITEM MI_LEFT, 112                //Left menu item
    HELPSUBITEM MI_CENTER, 113              //Center menu item
    HELPSUBITEM MI_RIGHT, 114               //Right menu item
    HELPSUBITEM MI_TEXT, 199                //Text menu item
  END                                     //

HELPSUBTABLE SUBTABLE_DIALOG                //Text dialog help subtable
  BEGIN                                   //
    HELPSUBITEM DID_ENTRY, 2 1              //Entry field <-> help ID 2 1
    HELPSUBITEM DID_OK, 2 2                 //OK button <-> help ID 2 2
    HELPSUBITEM DID_CANCEL, 2 3             //OK button <-> help ID 2 3
  END                                     //
```

WND_HELLO is a static text control ID and MI_* are menu item IDs. Each of these IDs is related to a panel ID. The main frame window ID, WND_MAIN, is also related to a panel ID. In this 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.

A pointer `help`, which points to the `IHelp` class, is added into the `AHelloWindow` class. An `AHelpHandler`, which is derived from `IHelpHandler`, overrides the member function `keysHelpId`, so that the correct Keys Help panel can be displayed when keys help is requested.

```
class IHelpHandler: public IHelpHandler
{
  protected:
   virtual Boolean
     keysHelpId(IEvent& evt);
};
```

The `keysHelpId` function is called when the user requests the keys help function. The default action is to set the event result to zero, which indicates to IPF to do nothing. In the example, this function is overridden and the result is set to the identity of the help window it wants IPF to display, in this case, the keys help panel.

```
Boolean AHelpHandler :: keysHelpId(IEvent& evt)
{
  evt.setResult(1   );
  return true;
}
```

The number `1` is the keys help ID defined in the AHELLOW5.IPF file.

The AHELLOW5.IPF file is compiled to produce AHELLOW5.HLP and added to the help window object, pointed to by `IHelp` in the example.

```
  help = new IHelpWindow(HELP_TABLE,this);
  help->addLibraries("AHELLOW5.HLP");
```

Use the `addLibraries` member function to add a library or list of libraries to the help window object, so that when you look for a help panel by panel ID, these libraries can be used (if multiple library names are specified, they should be separated by a blank space).

DBCS and NLS Support

The User Interface Class Library provides double-byte character set (DBCS) and national language support (NLS). You can use one source file for your application code and provide DBCS and NLS support by using separate resource files for the languages you support. The benefits of the approach include the following:

- The application is easy to maintain, because a single version of the application code is used. This reduces the cost of maintaining your code.
- The application is easy to upgrade, because only the source code is upgraded and then linked with different DLLs to generate different language versions. This reduces the time and cost of upgrading your code, because different language versions can be generated at the same time.

Because message strings are defined in resource files, they can be translated easily to another language without changes to source code. These resource files are linked to resource DLLs. From the command line, users can specify the parameter of the language they want to use so that applications link the correct resource DLLs at run time and generate the specified language messages. In Version 6 of the Hello World application, if you specify the parameter "/P" on the command line, the application links the Portuguese resource DLL file and generates the messages in Portuguese.

The canvas classes provide a window to control the position and size of message strings when the font changes and child windows are resized dynamically. The canvas classes also allow you to use different resource DLLs, thus simplifying the task of enabling your applications for DBCS and NLS support.

The following suggestions will assist you in creating DBCS-enabled applictions:

- The `IKeyboardEvent` class, which provides all the keyboard action event information, includes the `mixedCharacter` member function, which returns the event's single-byte or double-byte character. This function is recommended for DBCS-enabled applications.

- String manipulation is DBCS-enabled. The `IString` class supports mixed strings that contain both 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 the character is single-byte or double-byte, and whether it is a valid DBCS first byte.
- The `IDBCSBuffer` class 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:
  - `isCharValid`

    The return value is true if and only if the character at the given index is in the set of valid characters.
  - `isDBCS1`

    The return value is true if and only if the byte at the given offset is the first byte of DBCS.
  - `isPrevDBCS`

    The return value is true if and only if the character preceding the one at the given offset is a DBCS character.
- When you create and manage the entry field control window, you can set the style to be `anyData`, which allows the input text to be a mixture of SBCS and DBCS characters. For DBCS-only data, set the style to be `dbcsData`. For SBCS-only data, set the style to be `sbcsData`.
- In the `IComboBox` class, which combines an entry field and a list box to form one control containing both entry field and list box features, the default data style is `anyData` This allows the input text to be a mixture of SBCS and DBCS characters. For pure double-byte text, set the data style to be `dbcsData`.

  In the `IFrameWindow` class, add the `appDBCSStatus` member function into the style to include a DBCS status area when the frame appears in a DBCS environment. Use the `shareParentDBCSStatus`

member function to share DBCS status control between a parent and child frame.

Part 3. Sample Applications

© Copyright IBM Corp. 1993

Chapter 9. Introduction to the Sample Applications

This section shows you how to build an application using the User Interface Class Library. It is not designed to teach you the details of C++ programming. If you are unfamiliar with the principles and aspects of C++ programming, consult the *IBM C/C++ Programming Guide* before continuing with this section.

The creation of this 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 is designed to show you a different aspect of the User Interface Class Library.

Running the Samples

Sample files for each version are provided with the User Interface Class Library product diskettes. Installing and using the samples will help you understand the classes more quickly. For your convenience, complete listings of each file that contains sample code for this application are included.

In addition, files are included to help you compile and link each version of this sample application. READMEn.TXT files, where n is the version number, are also included with complete instructions for compiling and linking each version.

You'll notice that many versions of the sample application create pointers to objects (new). For simplicity, we do not always show object cleanup in our samples. When your applications create pointers to objects, the objects are not destroyed unless your application deletes them. Therefore, it is up to your application to use the C++ `delete` statement or to specify `setAutoDeleteObject` on your window objects to free the used memory when an object is no longer needed.

© Copyright IBM Corp. 1993

Conventions Overview

The User Interface Class Library uses a few conventions to enhance the usability and readability of the code. Here are two that will help you as you learn how to create an application. See Appendix B, “Class Library Conventions” on page 269 for information about other conventions used by the User Interface Class Library.

- Class names all begin with a capital letter. For example, all classes belonging to the User Interface 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 (e.g. `AHelloWindow`) for Application defined classes. This convention will help distinguish application classes from the classes that belong to the User Interface Class Library. You may find this standard useful, as well, to help you distinguish classes that you create from those supplied by the class library.

- Member functions begin with a lower case 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`.

Other Conventions Used in the Sample Code

Each version of the sample application builds upon the previous version in terms of complexity and functions provided. A version indicator (for example, `v2` or `v4`) appears in the sample code comments to indicate which statements were added to enhance the previous version. The following example illustrates this convention:

```
#include <istattxt.hpp>              //IStaticText Class
#include <iinfoa.hpp>               //IInfoArea Class                v2
#include <imenubar.hpp>              //IMenuBar Class                v3
#include <ifont.hpp>                //IFont                        v3
#include <istring.hpp>               //IString Class                 v4
#include <isetcv.hpp>                //ISetCanvas Class              v4
```

Chapter 10. Class Library Applications

To create a User Interface Class Library application, first you need to know which files to create and what goes in them. The following list describes the minimum files required for an application. Typically, the name of each file is the same; only the extensions differ.

filename.CPP Contains the primary C++ code for your application.

filename.HPP Contains the declaration of any class or classes that you create. You can put each class in a separate .HPP file or all classes in one file. If your classes are used in only one .CPP file, they can be declared in that .CPP file instead.

Optionally, you may want to create the following files:

filename.RC (and associated resources)
Application resource file. Used when the application requires data, such as text strings or bit maps, from an external source. Examples of external sources include .BMP, .ICO and .DLG files.

filename.H Defines constants used in a resource (.RC) file.

filename.DEF Module definition file. Contains information that defines your application for the linker.

filename.MAK Contains information to compile and link your application.

Structure of User Interface Class Library Applications

User Interface Class Library applications are written using the C++ programming language. These files have the following structure:

1. `#include` statements

   Insert `#include` statements at the beginning of the file to specify other files that contain information that your application will require. Typical `#include` statements are:

   - `#include <Ixxxxx.HPP>`

© Copyright IBM Corp. 1993

Includes the header file that contains information about an User Interface Class Library class that your application uses. The header file for each class you use must be included. All User Interface Class Library classes with a global scope begin with the letter "I." Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for an appendix that contains cross-reference tables for header files and the classes they contain.

Note: For faster compiles, surround `#include` statements with `#define` and `#endif` statements, as follows:

```
#ifndef _IXXXX_
  #include <IXXXX.HPP>
#endif
```

where `IXXXX` is the name of the class library header file being included (without the .HPP extension).

- `#include "xxxxx.HPP"`

  Represents the inclusion of a header file for a class that you have created. You must include header files for classes that you create if your application uses those classes. See "Creating Your Own Classes" on page 143 for more information.

- `#include "xxxxx.H"`

  Includes the file that defines your constants.

2. Main procedure to define the application entry point

   Normally, the main procedure creates the application window. The windows are then displayed and event processing is started for the application. This is can be done by using the `run` member function within the `ICurrentApplication` class. See "Run and Exit" on page 18 for more information.

3. Constructor for the application window

   The `IFrameWindow` class is used to construct the application window. This class can be called directly from your application, or it can be subclassed within a class of your own creation.

Once the application window is constructed, other classes can be called to insert controls and dialogs into the window, handle mouse and keyboard events, and so forth. The rest is up to you.

Creating Your Own Classes

Most applications will require new classes to be created. Most new classes can be derived from an existing base class. For developers familiar with other object oriented programming languages, a derived class is the C++ term for a subclass and a base class is the C++ term for a super class. Classes are derived from a base class to inherit implementation details and/or to be substitutes for the base class. When a derived class is just using the implementation details from a base class, private or protected inheritance should be used. When a derived class is a substitute for the base class, the developer should declare the base class using public inheritance. The following table provides a starting point to determine the base class to use:

New Function	Base Class
Adding a new dialog window	An IFrameWindow Class
Adding primary or secondary window	An IFrameWindow Class
Changing behavior of a window	An IHandler Class
Adding a new event	An IEvent Class
Adding a new control	An IControl Class or ITextControl
Adding a new canvas class	An ICanvas Class
Adding a new data type	An IBase or IVBase Class
Adding a new attribute	An IBase or IVBase Class
Adding a new cursor	An IVBase
Adding a new style	An IBitFlag Class
Adding a new settings	An IBase Class
Adding a new exception	An IException Class

Chapter 11. A Simple Application with a Main Window

Version 1 of the Hello World sample application creates a main window and inserts a text string into it using the static text control. In doing so, it shows you how to:

- Create the main window using the `IFrameWindow` class
- Create a static text control
- Put a text string into the control
- Set the static text control as the client window
- Run the application

The main window for Version 1 of the application looks like this:

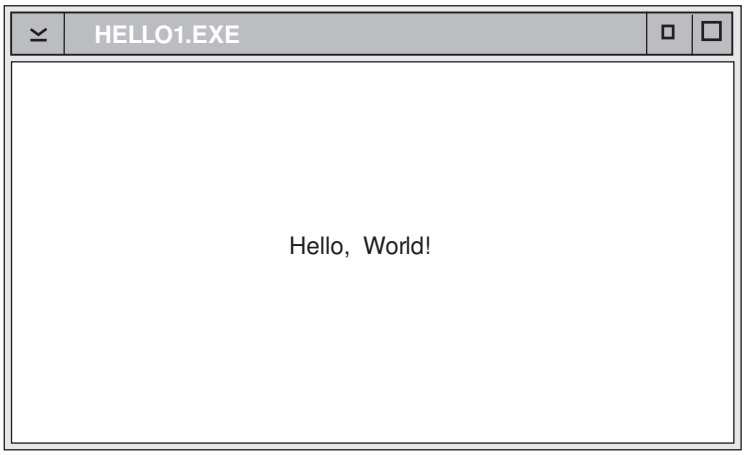


Figure 35. Version 1 of the Hello World Application

© Copyright IBM Corp. 1993

Version 1 Window Parent Relationship Diagram

Figure 36 shows the relationships between the objects built for Version 1 of the Hello World application:

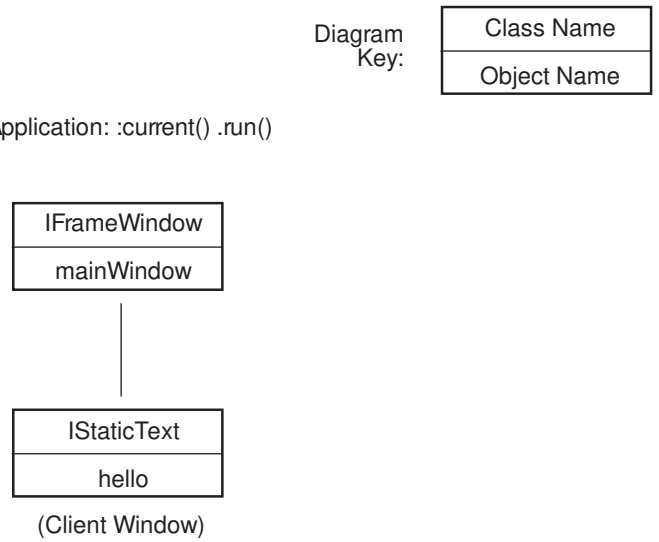


Figure 36. Window Parent Relationship Diagram

As the diagram shows, Version 1 of the Hello World application creates two objects: a main window and a static text control. The `mainWindow` object is the main window of the Hello World application.

The `hello` object, a static text control, is an instance of the `IStaticText` class. The phrase "(Client Window)" indicates that the static text control is displayed in the main window's client area. In this case, the *client area* is that part of the primary or main window inside the borders and below the title bar. In general, all space not used by the frame and its extensions belongs to the client area.

Version 1 Files

The AHELLOW1.CPP file contains the source code for the main procedure. The tasks performed by this code are described in “Tasks Performed by Version 1” on page 148 and its related sections.

AHELLOW1.CPP Source code for the main procedure.

AHELLOW1.DEF Module definition file for HELLO1.EXE.

The Source Code File

AHELLOW1.CPP contains the source code used for Version 1. Here is a listing of the source code:

```
                                  //Include IBM UI class headers:
#include <iapp.hpp>                   //IApplication Class
#include <istattxt.hpp>                //IStaticText Class
#include <iframe.hpp>                  //IFrameWindow Class Header

//*************************************************************************
// main  - Application entry point                                        *
//*************************************************************************
void main()                           //Main procedure with no parameters
{
  IFrameWindow * mainWindow=new         //Create our main window on the desktop
    IFrameWindow( x1   );              //  Pass in our Window ID

  IStaticText * hello=new IStaticText(  //Create static text control with
     x1 1 , mainWindow, mainWindow);    //  mainWindow as parent & owner
  hello->setText("Hello, World!");      //Set text in Static Text Control
  hello->setAlignment(                 //Set Alignment to Center in both
    IStaticText::centerCenter);         //  directions

  mainWindow->setClient(hello);         //Set hello control as Client Window
  mainWindow->setFocus();              //Set focus to main window
  mainWindow->show();                  //Set to show main window

  IApplication::current().run();        //Get the current application and
                                  // run it
} /* end main */
```

The Module Definition File

A module definition file may be created in order to define certain aspects of the application to the linker. See “The Module Definition File Format” on page 153 for information about the format of this file.

AHELLOW1.DEF, the module definition file we created for Version 1, contains the following:

```
NAME    HELLO1     WINDOWAPI

DESCRIPTION 'Hello World Sample C++ Program - Version 1'

CODE    LOADONCALL MOVEABLE
DATA    MOVEABLE   MULTIPLE

HEAPSIZE  16
STACKSIZE 65
```

Tasks Performed by Version 1

The following sections describe each of the tasks performed by Version 1 of the Hello World application. These tasks are:

- Creating the main window
- Creating a static text control
- Putting text in the static text control
- Aligning text within the static text control
- Setting the static text control as the client window
- Setting the focus and showing the main window
- Getting the current application and running it

Creating the Main Window

The first task is to create the main window for the application. The main window will be an instance of the `IFrameWindow` class. To make this class available, the application must include the IFRAME.HPP library header file, as follows:

```
#include <iframe.hpp>                //IFrameWindow Class Header
```

Now that the `IFrameWindow` class is available, a variable, in this case `mainWindow`, can be defined as a pointer to a new instance of this class, thus creating the main window of the application:

```
IFrameWindow * mainWindow=new       //Create our main window on the desktop
  IFrameWindow( x1   );            //  Pass in our Window ID
```

The hexadecimal value `x1` is assigned as the window ID.

Creating a Static Text Control for Version 1

The next task is to create a static text control for the “Hello, World!” text string. Since this control will be an instance of the `IStaticText` class, another library header file, ISTATTXT.HPP, must be included:

```
#include <istattxt.hpp>              //IStaticText Class
```

Now, another variable, in this case *hello*, can be defined as a pointer to a new instance of the `IStaticText` class, which creates a static text control:

```
IStaticText * hello=new IStaticText(  //Create static text control with
   x1 1 , mainWindow, mainWindow);    //  mainWindow as parent & owner
```

The hexadecimal value `x1 1` is assigned as the control ID.

The parameter that follows the hexadecimal value identifies the parent of the static text control, represented by the *mainWindow* variable. This is done so the static text control will be positioned in relation to the main window and displayed on top of the main window.

The last parameter identifies the main window as the owner of the static text control. Controls notify their owner windows when significant 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 will be reported to the main window, which is specified as the owner. In Version 1, no actions can be performed on the static text control, but that will change in Versions 2 through 6.

Putting Text in the Static Text Control

Now that the static text control has been created, it can be given a static text string. The `IStaticText` class is derived from the `ITextControl` class, and thus inherits its functions. One of those functions, `setText`, is used here to define the text string for the static text control:

```
hello->setText("Hello, World!");     //Set text in Static Text Control
```

Aligning Text within the Static Text Control

Next, the `setAlignment` function of the `IStaticText` class is used to align the text string in the static text control. In this case, it is centered both horizontally and vertically.

```
hello->setAlignment(               //Set the alignment to center both
  IStaticText::centerCenter);       // horizontally and vertically
```

If the text string had not been aligned, it would have been placed in the upper left corner of the client area (aligned left horizontally and at the top of the window vertically) by default.

Setting the Control as the Client Window

The next task is to designate the static text control as the frame's client window so the "Hello, World!" text string can be displayed in the main window's client area. This is done by using the `setClient` function of the `IFrameWindow` class:

```
mainWindow->setClient(hello);       //Set hello control as Client Window
```

The frame's *client window* is essentially the window corresponding to the client area, which is the rectangular portion of the frame window not occupied by the other frame controls (title bar, window border, minimize and maximize buttons, and so forth). 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 main window is resized, the client area (static text control in this example) grows or shrinks but the frame and its extensions remain the same size.

Setting the Focus and Showing the Main Window

Only two tasks remain in writing this application:

- Designating the main window as the active window
- Allowing the main window to be displayed when the application is run

These tasks are done by using the `setFocus` and `show` functions:

```
mainWindow->setFocus();             //Set focus to main window
mainWindow->show();                //Show main window
```

The `setFocus` and `show` functions are inherited from the `IWindow` class, as shown in the following class hierarchy:

```
IWindow
  |
  └─IFrameWindow
```

The User Interface Class Library allows classes that are used in an application to inherit functions from the classes from which they are derived without the application having to include those classes. Therefore, since `IFrameWindow` is derived from `IWindow`, the library header file that contains the declaration for the `IWindow` class does not need to be included in this application for its functions to be available.

Running the Application

The last task is to get the main window displayed and start the user interface event processing for the application. This involves getting and dispatching window events until the application is terminated. This sample application accomplishes the task using the function `ICurrentApplication::run()`. This requires using member functions that belong to the `IApplication` and `ICurrentApplication` classes. Therefore, another library header file, IAPP.HPP, must be included:

```
#include <iapp.hpp>                 // IApplication class
```

Hello World — Version 1

The `current` member function of the `IApplication` class returns the current application, in this case the Hello World application, as an instance of the `ICurrentApplication` class. Once this is done, the `ICurrentApplication` class's `run` function can be used to display the main window and start event processing for this application.

```
IApplication::current().run();       //Get the current application and
                                     // run it
```

Compiling and Linking Version 1

Figure 37 shows the files that were used to create Version 1 of the Hello World application, their relationship to each other, and the order in which they are compiled and linked. File names are shown in uppercase letters; program names are shown in lowercase letters.

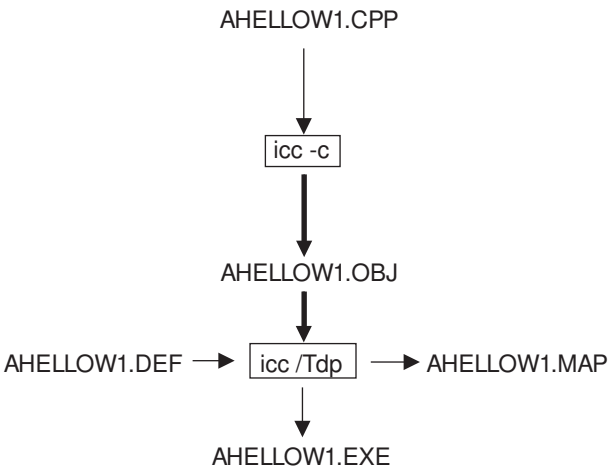


Figure 37. Compiling and Linking Version 1 of Hello World Application

The Module Definition File Format

A module definition file may be created in order to define certain aspects of the application to the linker. This file provides the following information.

NAME The application name and type.

DESCRIPTION A short description of the application.

CODE Information about the attributes for the code segment:

- **LOADONCALL** Specifies that the code segment is loaded when called.
- **MOVEABLE** Specifies that the code segment is moveable.

DATA Information about the data segment:

- **MOVEABLE** Specifies that the data segment is moveable.
- **MULTIPLE** Causes a data segment to be created for each instance of the executable code.

HEAPSIZE A user-defined minimum heap size, rather than letting the linker determine the minimum heap size.

STACKSIZE A user-defined stack size, rather than letting the linker determine the stack size.

See “The Module Definition File” on page 148 for the module definition file used by this application.

Hello World — Version 1

Chapter 12. Adding a Resource File and Frame Extensions

This chapter 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 PM 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 from a resource file
- Set the window title and system menu icon from a resource file
- Create and set the information area below the client area

The window for Version 2 of the Hello World application looks like this:

© Copyright IBM Corp. 1993

Hello World — Version 2

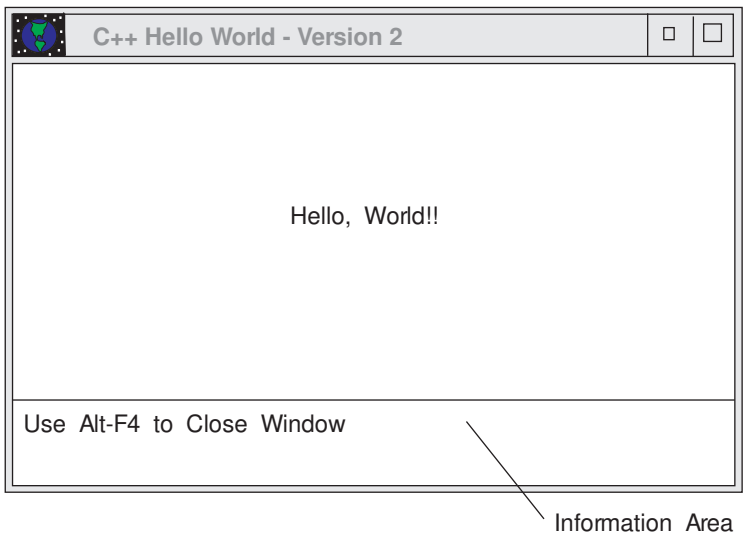


Figure 38. Version 2 of Hello World Application

Version 2 Window Parent Relationship Diagram

Figure 39 on page 157 shows the relationship between the objects built for Version 2 of the Hello World application:

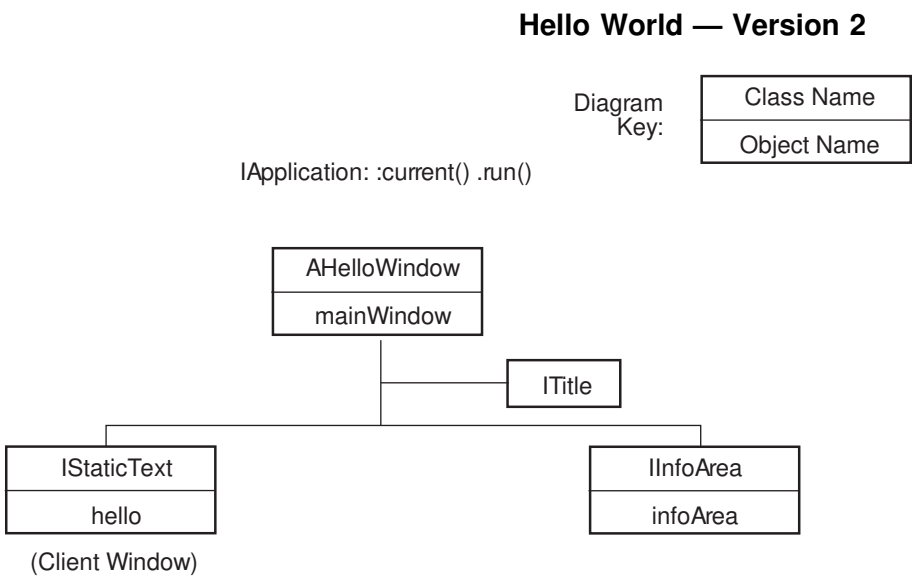


Figure 39. Window Parent Relationship Diagram

For Version 2, the `mainWindow` object is an instance of the `AHelloWindow` class, a subclass that is created for Version 2 and derived from the `IFrameWindow` class.

The `hello` object is the same as in Version 1.

In addition to the `mainWindow` and `hello` objects, Version 2 provides:

- An instance of the `ITitle` class, for the window title
- An `infoArea` object, which is an instance of the `IInfoArea` class that is used to display text in an information area in the main window.

Version 2 Files

The following files contain the code used to create Version 2:

AHELLOW2.CPP Source code for the main procedure and window constructor.

AHELLOW2.HPP Header file for the `AHellowWindow` class.

AHELLOW2.H Constant definition file for HELLO2.EXE.

AHELLOW2.RC Resource file for HELLO2.EXE.

AHELLOW2.ICO Icon file for HELLO2.EXE.

AHELLOW2.DEF Module definition file for HELLO2.EXE.

The Primary Source Code File

The AHELLOW2.CPP file contains the source code for the main procedure and the window constructor. If lines 79-80 contain a v2 or a period, then this source line was modified or added in this version. The tasks performed by this code are described in “Tasks Performed by Version 2” on page 164 and its related sections.

```
                                  //Include the IBM UI class headers:
#include <iapp.hpp>                   //IApplication Class
#include <istattxt.hpp>                //IStaticText Class
#include <iinfoa.hpp>                  //IInfoArea Class                v2

#include "ahellow2.hpp"                //Include the AHelloWindow class     v2
                                  //  header                        v2
#include "ahellow2.h"                  //Include our symbolic definitions   v2

//*************************************************************************
// main  - Application entry point                                        *
//*************************************************************************
void main()                         //Main procedure with no parameters
{
  AHelloWindow mainWindow (WND_MAIN);   //Create our main window on the
                                  // desktop
  IApplication::current().run();        //Get the current application and
                                  // run it
} /* end main */
```

Hello World — Version 2

```
//*************************************************************************
// AHelloWindow :: AHelloWindow - Constructor for our main window        *
//*************************************************************************
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow (                  //Call the IFrameWindow constructor
    IFrameWindow::defaultStyle()       //  using the default style, plus    v2
    | IFrameWindow::minimizedIcon,     //  get minimized icon from RC file   v2
    windowId)                       //  and set the main window ID
{
  hello=new IStaticText(WND_HELLO,     //Create a static text control
    this, this);                    //  Pass in this AHelloWindow as the
                                   //  parent and owner of the control
  hello->setText(STR_HELLO);          //Set text in the static text control v2
                                   // from the RC file              v2
  hello->setAlignment(              //Set the alignment to center both
    IStaticText::centerCenter);       // horizontally and vertically
  setClient(hello);                 //Set the static text control as the
                                   // client window
  infoArea=new IInfoArea(this);       //Create the information area       v2
  infoArea->setInactiveText(STR_INFO);  //Set information area text from RC   v2

  sizeTo(ISize(4  ,3  ));            //Set the pixel size of main window  v2
  setFocus();                      //Set the focus to the main window
  show();                          //Show the main window

} /* end AHelloWindow :: AHelloWindow(...) */
```

The AHelloWindow Class Header File

AHELLOW2.HPP is not an User Interface Class Library header file. Instead, it is the type of header file that you would create for a class of your own. In this case, it contains the class definition and interface specifications for the `AHelloWindow` class, a subclass of `IFrameWindow` that we created specifically for this application. Here is the source listing for AHELLOW2.HPP:

```
#ifndef AHELLOWINDOW_HPP
#define AHELLOWINDOW_HPP

#include <iframe.hpp>                //Include the IFrameWindow class
                                    // header

//****************************************************************************
// Class:   AHelloWindow                                                    *
//                                                                         *
// Purpose: Main window for the C++ Hello World sample application.       *
//         It is a subclass of IFrameWindow.                                *
//                                                                         *
//****************************************************************************
class AHelloWindow : public IFrameWindow
{
  public:                           //Define the public Information
    AHelloWindow(unsigned long windowId);//Constructor for this class

  private:                           //Define the private Information
    IStaticText   * hello;             //Define a Static Text Control to
                                     //  keep the "Hello, World" text
                                     //  and as the client window
    IInfoArea     * infoArea;          //Define an Information Area        v2
                                     //  Control to create an information   .
                                     //  area beneath the client area     v2
};
#endif
```

The Constants Definition File

AHELLOW2.H contains our constant definitions for this application. These constants and their definitions provide the IDs for the application main window, controls and text strings. They are required because, in this version of the application, the text strings are being pulled in from a resource file. The constants and their definitions are shown in the following code:

```
#ifndef AHELLOWINDOW_H
#define AHELLOWINDOW_H
//****************************************************************************
// window IDs - Used by IWindow constructors, such as IStaticText and        *
//            AHelloWindow.                                                  *
//****************************************************************************
#define WND_MAIN         x1           //Main window ID

#define WND_HELLO        x1 1         //Hello World window ID
#define WND_INFO         x1 12        //Information area ID              v2

//*************************************************************************  v2
// string IDs - Used to relate resources to IStaticText and ITitle.       *   .
//*************************************************************************  v2
#define STR_HELLO        x12          //Hello World string ID             v2
#define STR_INFO         x122         //Information area string ID         v2

#endif
```

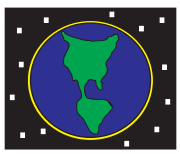
The Resource File

Version 2 of the Hello World application provides a resource file, AHELLOW2.RC. This resource file assigns an icon and three text strings to the constants defined in the AHELLOW2.H file shown in “The Constants Definition File” on page 161. AHELLOW2.H is included in this resource file so the icon and text strings can be associated with the appropriate IDs. Here is the code used in the AHELLOW2.RC file:

```
#include "ahellow2.h"                    //Symbolic definitions      v2

//************************************************************************  v2
// Icon and bit-map resources                                        *   .
//  Symbolic Name (ID) <-> Icon File Name                              *   .
//************************************************************************   .
ICON WND_MAIN ahellow2.ico                //Title bar icon (use same ID)v2

//************************************************************************  v2
// string resources - Used by the IStaticText and ITitle classes         *   .
//  Symbolic Name (ID) <-> Text String                                 *   .
//************************************************************************  v2
STRINGTABLE
  BEGIN
    STR_HELLO,  "Hello, World!!"            //Hello World text string    v2
    WND_MAIN,   "C++ Hello World - Version 2"  //Title bar text (main ID)   v2
    STR_INFO,   "Use Alt-F4 to close window"   //Information area text      v2
  END
```

The Icon File

AHELLOW2.ICO is used as both the title bar icon and the icon that is displayed when the application is minimized. We cannot provide a listing for the AHELLOW2.ICO file, but this is how the icon appears when minimized:



Hello World Icon

Figure 40. Hello World Icon

AHELLOW2.DEF

The AHELLOW2.DEF file is required for the same reasons that AHELLOW1.DEF was needed for Version 1. See “The Module Definition File” on page 148 if you need to review the reasons for creating a .DEF file.

The only difference between the two .DEF files used in Version 1 and Version 2 is the change in the version number.

```
NAME    HELLO2    WINDOWAPI

DESCRIPTION 'Hello World Sample C++ Program - Version 2'

CODE    LOADONCALL MOVEABLE
DATA    MOVEABLE   MULTIPLE

HEAPSIZE  8192
STACKSIZE 256
```

Advantages of the C++ File Structure

In Version 1 of the Hello World application, all of the source code was intentionally put in the AHELLOW1.CPP file to make that version of the application very simple. However, for Version 2, the source code has been distributed among a variety of files in order to show the flexibility and versatility that you can achieve by structuring your applications this way.

First of all, the `AHelloWindow` class, the subclass of `IFrameWindow` that was referred to in the preceding section, 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 over again with other applications and implemented in different ways. If the class definition or interface specifications change, they change in only one place, the header file.

Similarly, the constant definitions file (AHELLOW2.H) allows IDs to be assigned to the windows and text strings in one place. Defining the constants in one place allows those constants to be used in a variety of places, such as the source code and the resource file, while keeping

their definitions in one place. Then, if a need to change the constant definitions arises, only the AHELLOW2.H file must be modified.

The advantage of placing the application's data in a resource file (AHELLOW2.RC) is that it allows all of the resources to be specified in one place. For example, finding and modifying text strings is much easier when they are all grouped in one place, rather than if you had to search through the source code for each one.

Tasks Performed by 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 are described again because they are done a little differently in Version 2. The tasks are:

- Creating the main window
- Getting the current application and running it
- Constructing the main window, which involves the following:
  - Creating a static text control
  - Setting a text string from a resource file
  - Putting a text string into a static text control
  - Aligning the text
  - Setting the static text control in the main window
  - Setting the window title and title bar icon from a resource file
  - Creating and setting the information area below the client area
  - Setting the focus to the main window and showing the main window

Creating the Main Window

One of the major differences between Version 1 of the Hello World application and Version 2 is the manner in which the main window is created. Version 1 simply creates an instance of the `IFrameWindow` class. However, Version 2 provides its own class, `AHelloWindow`, for creating 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 line to make the derivation of the `AHelloWindow` class from the `IFrameWindow` class possible:

```
//<in ahellow2.hpp>
#include <iframe.hpp>                  //Include the IFrameWindow class
                                     // header
```

Note: See “Version 2 Files” on page 158 to learn about reasons for putting class definitions and interface specifications in a header file.

The AHELLOW2.CPP file, which contains most of the source code for the application, includes the AHELLOW2.HPP header file in order to have access to the `AHelloWindow` class:

```
// <in AHELLOW2.CPP>
#include "ahellow2.hpp"               //Include the AHelloWindow class
                                    // header
```

The following line in the AHELLOW2.CPP file creates the main window by using the `AHelloWindow` class constructor:

```
// <in AHELLOW2.CPP>
  AHelloWindow mainWindow (WND_MAIN);   //Create the main window on
                                      // the desktop
```

In Version 1, the main window was given a hexadecimal value of `x1` as its window ID when the main window was created. The same value is used for the window ID of the main window in Version 2. However, instead of specifying that value in the primary source code file, Version 2 uses a constant, WND_MAIN, which is defined in the AHELLOW2.H file, as follows:

```
//<in ahellow2.h>
#define WND_MAIN          x1           //Main window ID
```

Note: See “Version 2 Files” on page 158 to learn about reasons for using a constants definition file.

In order to have access to this definition, the primary source code file, AHELLOW2.CPP, must include the AHELLOW2.H file, as follows:

```
// <in AHELLOW2.CPP>
#include "ahellow2.h"                //Include definitions of the constants
```

Getting the Current Application and Running It

When the main window is constructed, the following line gets the current application and runs it.

```
IApplication::current().run();       //Get the current application and
                                  // run it
```

See “Running the Application” on page 151 for a more detailed explanation.

Once the main window has been created, it next must be constructed. The following sections explain how this is done.

Constructing the Main Window

Version 2 of the Hello World application constructs the main window using the `AHelloWindow` class. Here is the class constructor as it is defined in the AHELLOW2.HPP header file:

```
// <in AHELLOW2.HPP>
AHelloWindow(unsigned long windowId); //Constructor for this class
```

In the primary source code file, Version 2 uses the following lines of code to construct the main window:

```
// <in AHELLOW2.CPP>
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow (                  //Call the IFrameWindow constructor
    IFrameWindow::defaultStyle()       //  using the default style, plus    v2
    | IFrameWindow::minimizedIcon,     //  get minimized icon from RC file   v2
    windowId)                       //  and set the main window ID
```

Two capabilities provided by the `IFrameWindow` class have been used here that were not used in Version 1:

- Setting the main window to the default style

  The `defaultStyle` function is inherited from the `IFrameWindow` class. It returns the current default style that your application is using for all frame windows. The current default style is either the original default style that is provided by the User Interface Class Library for frame windows, or a new default style that has been established by using the `setDefaultStyle` function.

  In this case, since the `setDefaultStyle` function has not been used, the current default style is the same as the original default style,

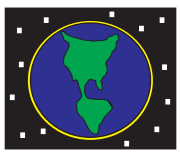
which provides a title bar, title bar icon, minimize button, maximize button, window border, window list, and an initial shell position for the window.

In the Hello World application, the text and icon for the title bar are specified in the resource file, AHELLOW2.RC, which is described in the following sections. The text string for the window title is included in the resource file, and the icon, AHELLOW2.ICO, is specified.

Refer to “Styles” on page 55 and to the *IBM C/C++ Tools: User Interface Class Library Reference* for more information about styles.

- Getting an icon that is used when the main window is minimized

  The `minimizedIcon` function is also inherited from the `IFrameWindow` class. This function allows your application to use an icon, contained in your .EXE file and specified in your resource file, to represent the application when it is minimized on the desktop. The Hello World application provides the AHELLOW2.ICO icon file for this purpose. The following figure shows how this icon appears when the main window is minimized:



Hello World Icon

Figure 41. Hello World Icon

Creating a Static Text Control for Version 2

Another difference between Version 1 and Version 2 is the means of creating a static text control to display a text string. In Version 1, this was done very simply by setting `hello` equal to a new instance of the `IStaticText` class, associating an ID with the control window (`x1 1`), and making the main window the parent and owner of the control, as follows:

```
// <in AHELLOW1.CPP>
IStaticText * hello=new IStaticText(  //Create static text control with
   x1 1 , mainWindow, mainWindow);    // mainWindow as parent and owner
```

Hello World — Version 2

In Version 2, however, this code is divided into separate parts and placed in different files. As shown in the following lines of code, `hello` is now declared in the `AHelloWindow` class:

```
// <in AHELLOW2.HPP>
  IStaticText   * hello;             //Define a Static Text Control to
                                    //  keep the "Hello, World" text
                                    //  as the client window
```

In the AHELLOW2.CPP file, `hello` is used to create a new instance of a static text control:

```
// <in AHELLOW2.CPP>
hello = new IStaticText(WND_HELLO,    //Create a static text control
  this, this);                     // Pass in this AHelloWindow  as the
                                  //  parent and owner of the control
```

The WND_HELLO constant provides the ID for the static text control. All Presentation Manager windows must have a unique ID, including controls. Therefore, the AHELLOW2.CPP file must include AHELLOW2.H, because that is where this constant is defined:

```
// <in AHELLOW2.CPP>
#include "ahellow2.h"               //Include our symbolic definitions   v2
```

With the AHELLOW2.H included, the ID is associated with the WND_HELLO constant:

```
// <in AHELLOW2.H>
#define WND_HELLO        x1 1        //Hello World window ID
```

The other two parameters (`this, this`) are used to pass in the main window (this instance of the `AHelloWindow` class) as the parent and owner of the static text control. See “Creating a Static Text Control for Version 1” on page 149 for information about parent and owner windows.

Setting a Text String from an RC File for the Static Text Control

Once the static text control is created, the next task is to set text in it. Version 2 of the Hello World application gets the text string from a resource file. To do this, it uses the `setText` function, which is inherited from the `ITextControl` class:

```
// <in AHELLOW2.CPP>
hello->setText(STR_HELLO);          //Set text in the static text control
                                  // from the resource file
```

The `setText` member function finds this constant in the AHELLOW2.RC resource file and puts it into the static text control:

```
// <in AHELLOW2.RC>
STR_HELLO,  "Hello, World!!"              //Hello World text string
```

As we noted earlier, each window, even a control, must have a numeric value assigned as its ID. The STR_HELLO constant is associated with a string ID, hexadecimal value `x12`, in the AHELLOW2.H constant definition file. The resource file includes the constant definition file, so this constant definition is available.

```
// <in AHELLOW2.H>
#define STR_HELLO        x12          //Hello World string ID
```

Aligning 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:

```
hello->setAlignment(               //Set the alignment to center in both
  IStaticText::centerCenter);       // directions
```

Setting the Control as the Client Window

The last task to perform for the static text control is to set it as the client window. See “Setting the Control as the Client Window” on page 150 for an explanation of client windows.

```
setClient(hello);                   //Set the static text control as the
                                  // client window
```

Creating an Information Area

The following code creates a new instance of an information area using the `IInfoArea` class. This class provides a frame extension at the bottom of the client area that shows information about the application.

```
infoArea=new IInfoArea(this);       //Create the information area       v2
```

Setting Information Area Text from the Resource File

Normally, the information shown in the information area pertains to the frame menu item at which the selection cursor is currently positioned. The information is taken from a resource string table. A different text string is displayed 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 the “inactive text”) that is displayed whenever no menu item is selected.

Version 2 sets 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 only difference is the `setInactiveText` function of the `IInfoArea` class is used instead of the `setText` function:

```
infoArea->setInactiveText(STR_INFO);  //Set information area text from RC   v2
```

The `setInactiveText` member function finds the STR_INFO constant in the AHELLOW2.RC resource file and puts it into the information area:

```
// <in AHELLOW2.RC>
STR_INFO,   "Use Alt-F4 to close window"   //Information area text      v2
```

The STR_INFO constant is associated with a string ID, hexadecimal value `x122` , in the AHELLOW2.H constant definition file. The resource file includes the constant definition file, so this constant definition is available.

```
// <in AHELLOW2.H>
#define STR_INFO          x122          //Information area string ID         v2
```

Setting the Size of the Main Window

In Version 1, the main window's default size was used when it was displayed. Version 2 shows you how to change the size:

```
sizeTo(ISize(4  ,3  ));             //Set the pixel size of main window   v2
```

This sets the size of the main window to 400 pixels wide by 300 pixels high.

Setting the Focus and Showing the Main Window

As in Version 1, the last two member functions used are `setFocus` and `show`. However, since the `AHelloWindow` class is the parent and owner of the main window, you only need to specify the function names:

```
setFocus();                        //Set the focus to the main window
show();                           //Show window
```

Compiling and Linking Version 2

Figure 42 on page 172 shows the files that were used to create Version 2 of the Hello World application, their relationship to each other, and the order in which they are compiled and linked. File names are shown in uppercase letters; program names are shown in lowercase letters.

Hello World — Version 2

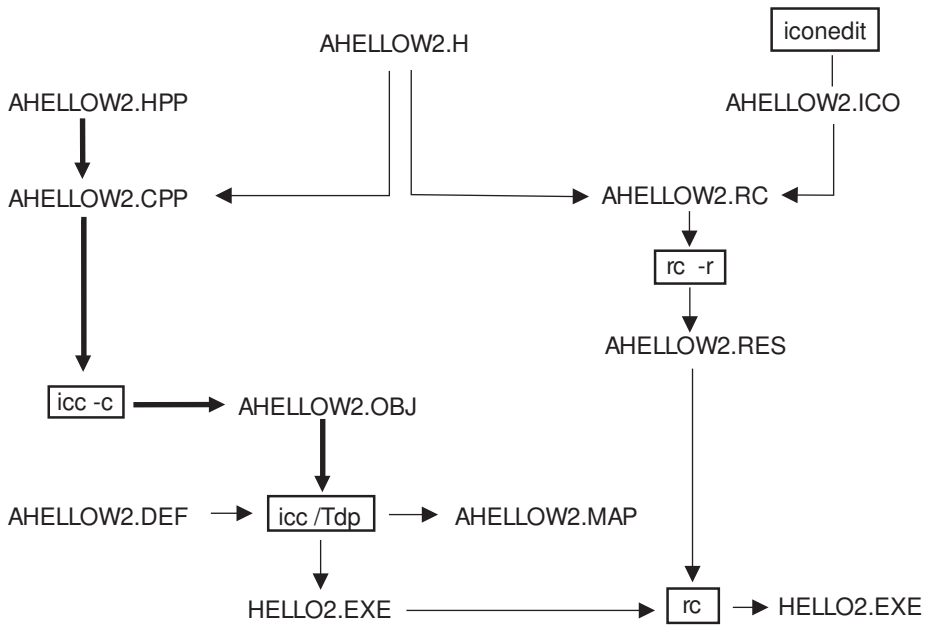


Figure 42. Compiling and Linking Version 2 of the Hello World Application

Chapter 13. Event Handling and Menu Bars

Version 3 provides a menu bar with an Alignment choice. By selecting this choice, the user can display a pull-down menu and align the “Hello, World!!!” text string 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:

- Add a menu bar to an application window
- Add command processing (event handling) to align a text string
- Place a check mark next to the selected pull-down menu choice
- Create a status line to show the status of the text string alignment
- Add string resources so the information area is updated when the user is selecting menu items

The window for Version 3 of the Hello World application looks like this:

© Copyright IBM Corp. 1993

Hello World — Version 3

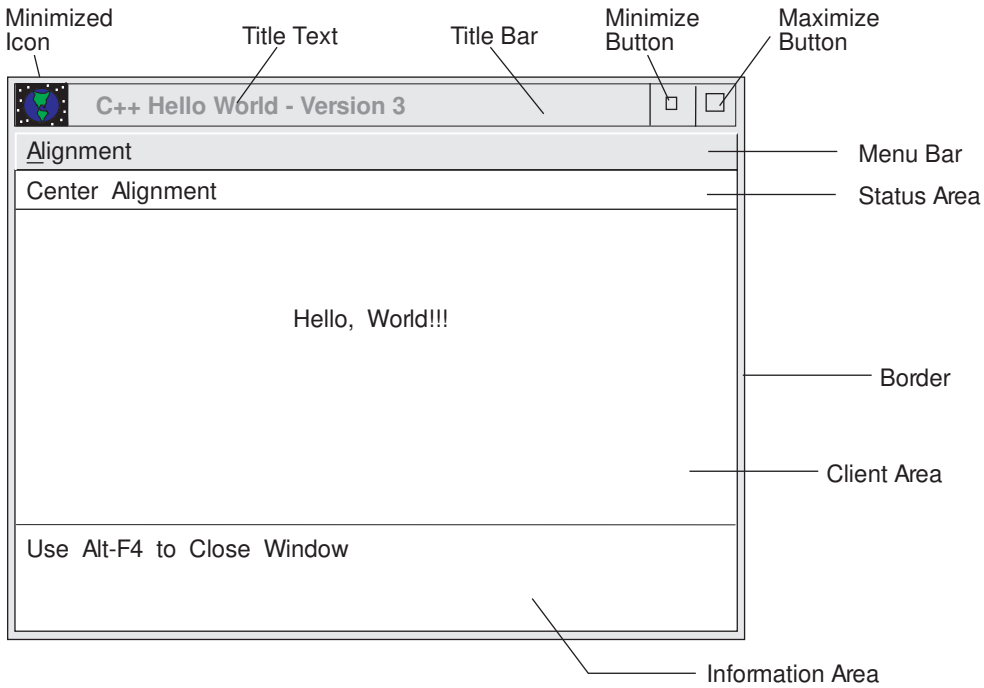


Figure 43. Version 3 of Hello World Application

Version 3 Window-Parent Relationship Diagram

Figure 44 on page 175 shows the relationship between the objects built for Version 3 of the Hello World application:

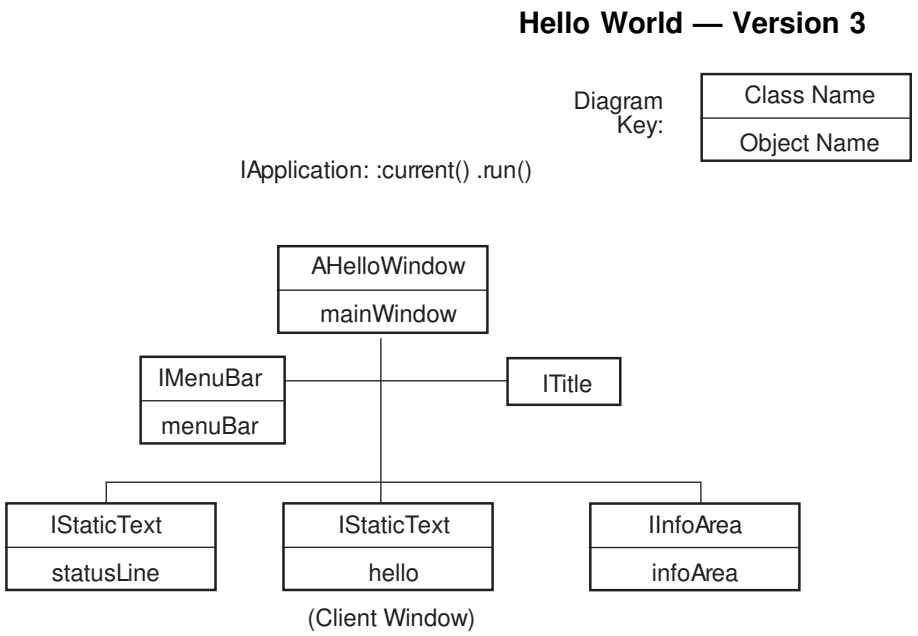


Figure 44. Window Parent Relationship Diagram

Version 3 Files

The following files contain the code used to create this window:

AHELLOW3.CPP Source code for the main procedure, main window constructor, and command processing.

AHELLOW3.HPP Header file for the `AHellowWindow` class.

AHELLOW3.H Constants definition file for HELLO3.EXE.

AHELLOW3.RC Resource file for HELLO3.EXE.

AHELLOW3.ICO Icon file for HELLO3.EXE.

AHELLOW3.DEF Module definition file for HELLO3.EXE.

The Primary Source Code File

The AHELLOW3.CPP file contains the source code for the main procedure, window constructor, and menu commands. The tasks performed by this code are described in “Tasks Performed by Version 3” on page 182 and its related sections.

Here is a listing of the primary source code:

```
                                //Include IBM UI class headers:
#include <iapp.hpp>                 //IApplication Class
#include <istattxt.hpp>              //IStaticText Class
#include <iinfoa.hpp>               //IInfoArea Class                v2
#include <imenubar.hpp>              //IMenuBar Class                v3
#include <ifont.hpp>                //IFont                        v3

#include "ahellow3.hpp"              //Include AHelloWindow Class headers  v2
#include "ahellow3.h"               //Include our Symbolic definitions   v2

//*************************************************************************
// main  - Application entry point                                      *
//*************************************************************************
void main()                         //Main Procedure with no parameters
{
  AHelloWindow mainWindow (WND_MAIN);   //Create our main window on the
                                // desktop
  IApplication::current().run();       //Get the current application and
                                // run it
} /* end main */

//*************************************************************************
// AHelloWindow :: AHelloWindow - Constructor for our main window        *
//*************************************************************************
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow (                  //Call IFrameWindow constructor      v2
    IFrameWindow::defaultStyle()       //  Use default plus              v2
    | IFrameWindow::minimizedIcon,     //  Get Minimized Icon from  RC file  v2
    windowId)                       //  Main Window ID
{
  hello=new IStaticText(WND_HELLO,     //Create Static Text Control
    this, this);                    //  Pass in myself as parent & owner
  hello->setText(STR_HELLO);          //Set text in Static Text Control    v2
  hello->setAlignment(               //Set Alignment to Center in both
    IStaticText::centerCenter);        //  directions
  setClient(hello);                 //Set hello control as Client Window

  infoArea=new IInfoArea(this);        //Create the information area       v2
  infoArea->setInactiveText(STR_INFO);  //Set information area text from RC   v2

  statusLine=new IStaticText          //Create Status Area using Static Text v3
```

```
   (WND_STATUS, this, this);          //  Window ID, Parent, Owner Parameters.
  statusLine->setText(STR_CENTER);     //Set Status Text to "Center" from Res .
  addExtension(statusLine,            //Add Status Line above the client    .
   IFrameWindow::aboveClient,         //  and specify the location          .
   IFont(statusLine).maxCharHeight()); //  and specify height              v3

  handleEventsFor(this);              //Set self as event handler (commands)v3
  menuBar=new IMenuBar(WND_MAIN, this); //Create Menu Bar for main window     .
  menuBar->checkItem(MI_CENTER);       //Place Check on Center Menu Item    v3

  sizeTo(ISize(4  ,3  ));             //Set the size of main window       v2
  setFocus();                        //Set focus to main window
  show();                           //Set to show main window

} /* end AHelloWindow :: AHelloWindow(...) */

//*************************************************************************  v3
// AHelloWindow :: command                                          *   .
//   Handle menu commands                                           *   .
//*************************************************************************   .
Boolean AHelloWindow :: command(ICommandEvent & cmdEvent)                // .
{                                                                  //v3
  switch (cmdEvent.commandId()) {      //Get command id                  v3

    case MI_CENTER:                  //Code to Process Center Command Item v3
      hello->setAlignment(            //Set alignment of hello text to      .
        IStaticText::centerCenter);    //  center-vertical, center-horizontal .
      statusLine->setText(STR_CENTER);  //Set Status Text to "Center" from Res .
      menuBar->checkItem(MI_CENTER);    //Place Check on Center Menu Item     .
      menuBar->uncheckItem(MI_LEFT);    //Uncheck Left Menu Item             .
      menuBar->uncheckItem(MI_RIGHT);   //Uncheck Right Menu Item            .
      return(true);                    //Return command processed          .
      break;                          //                               v3

    case MI_LEFT:                    //Code to Process Left Command Item   v3
      hello->setAlignment(            //Set alignment of hello text to      .
        IStaticText::centerLeft);      //  center-vertical, left-horizontal   .
      statusLine->setText(STR_LEFT);    //Set Status Text to "Left" from Res   .
      menuBar->uncheckItem(MI_CENTER);  //Uncheck Center Menu Item           .
      menuBar->checkItem(MI_LEFT);      //Place Check on Left Menu Item       .
      menuBar->uncheckItem(MI_RIGHT);   //Uncheck Right Menu Item            .
      return(true);                    //Return command processed          .
      break;                          //                               v3

    case MI_RIGHT:                   //Code to Process Right Command Item  v3
      hello->setAlignment(            //Set alignment of hello text to      .
        IStaticText::centerRight);     //  center-vertical, right-horizontal  .
      statusLine->setText(STR_RIGHT);   //Set Status Text to "Right" from Res  .
      menuBar->uncheckItem(MI_CENTER);  //Uncheck Center Menu Item           .
      menuBar->uncheckItem(MI_LEFT);    //Uncheck Left Menu Item             .
      menuBar->checkItem(MI_RIGHT);     //Place Check on Right Menu Item      .
```

```
    return(true);                 //Return command processed          .
    break;                       //                                v3

  } /* end switch */               //                                 v3

  return(false);                  //Return command not processed      v3
} /* end HelloWindow :: command(...) */                              //v3
```

The AHelloWindow Class Header File

Like AHELLOW2.HPP, AHELLOW3..HPP contains the class definition and interface specifications for the `AHelloWindow` class, with a few modifications for Version 3. Here is the source listing for AHELLOW3.HPP:

```
#ifndef AHELLOWINDOW_HPP
#define AHELLOWINDOW_HPP

#include <iframe.hpp>         //Include IFrameWindow Class Header
#include <icmdhdr.hpp>        //Include ICommandEvent & ICommandHandler    v3

//****************************************************************************
// Class:   AHelloWindow                                                    *
//                                                                        *
// Purpose: Main Window for C++ Hello World sample application           *
//         It is a subclass of IFrameWindow & ICommandHandler           *  v3
//                                                                        *
//****************************************************************************
class AHelloWindow : public IFrameWindow,
                 public ICommandHandler                          //v3
{
  public:                          //Define the Public Information
    AHelloWindow(unsigned long windowId); //Constructor for this class

  protected:                        //Define Protected Member          v3
    Boolean command(ICommandEvent& cmdEvent);                         //v3

  private:                         //Define Private Information
    IStaticText   * hello;           //Hello contains "Hello, World" text
    IInfoArea    * infoArea;         //Define an Information Area        v2
                                  //  Control to create an information   .
                                  //  area beneath the client area     v2
    IStaticText   * statusLine;       //Status Line at top of client window v3
    IMenuBar     * menuBar;          //Define Menu Bar                  v3
};
#endif
```

The Constants Definition File

AHELLOW3.H contains the constant definitions for this application. These constants and their definitions, which provide the IDs for the application window components, are shown in the following code:

```
#ifndef AHELLOWINDOW_H
#define AHELLOWINDOW_H
//****************************************************************************
// window ids - used by IWindow constructors (eg IStaticText, AHelloWindow)*
//****************************************************************************
#define WND_MAIN         x1           //Main Window ID

#define WND_HELLO        x1 1         //Hello World Window ID
#define WND_INFO         x1 12        //Information Area                v2
#define WND_STATUS       x1 11        //Status Line Window ID             v3

//****************************************************************************  v2
// string ids - used to relate resources to IStaticText and ITitle        *   .
//****************************************************************************  v2
#define STR_HELLO        x12          //Hello World String ID             v2
#define STR_INFO         x122         //Info String ID                  v2
#define STR_CENTER       x123         //Center Alignment Status String ID   v3
#define STR_LEFT         x1231        //Left Alignment Status String ID     .
#define STR_RIGHT        x1232        //Right Alignment Status String ID    v3

//****************************************************************************  v3
// menu ids - used on relate command ID to Menu Items and Function Keys    *   .
//****************************************************************************   .
#define MI_ALIGNMENT      x15          //Alignment Menu ID                 .
#define MI_CENTER        x15 1        //Center Menu ID                   .
#define MI_LEFT          x15 2        //Left Menu ID                    .
#define MI_RIGHT         x15 3        //Right Menu ID                  v3

#endif
```

For Version 3, the constants 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 of the Hello World application provides a resource file, AHELLOW3.RC. This resource file associates an icon and several text strings with the symbols defined in the AHELLOW3.H file shown in “The Constants Definition File” on page 179. It also contains the text strings for the menu bar. AHELLOW3.H is included in this resource file so the icon and text strings can be associated with the appropriate IDs. Here is the code used in the AHELLOW3.RC file:

```
#include "ahellow3.h"                      //Symbolic Definitions      v2

//*************************************************************************  v2
// icon and bitmap resources                                          *   .
//  Symbolic Name (ID) <-> icon filename                               *   .
//*************************************************************************   .
ICON WND_MAIN ahellow3.ico                //Title Bar Icon (use same id)v2

//*************************************************************************  v2
// string resources - used by IStaticText & ITitle Classes                *   .
//  Symbolic Name (ID) <-> Text String                                 *   .
//*************************************************************************  v2
STRINGTABLE
  BEGIN
    STR_HELLO,  "Hello, World!!!"           //Hello World String        v3
    WND_MAIN,   "C++ Hello World - Version 3"  //Title Bar String (main id)  v3
    STR_INFO,   "Use Alt-F4 to Close Window"   //Information Area String    v2
    MI_ALIGNMENT,"Alignment Menu"           //InfoArea - Alignment Menu   v3
    MI_CENTER,  "Set Center Alignment"       //InfoArea - Center Menu      .
    MI_LEFT,    "Set Left Alignment"        //InfoArea - Left Menu        .
    MI_RIGHT,   "Set Right Alignment"        //InfoArea - Right Menu      v3
    STR_CENTER, "Center Alignment"           //Status Line Text - Center   v3
    STR_LEFT,   "Left Alignment"            //Status Line Text - Left     .
    STR_RIGHT,  "Right Alignment"           //Status Line Text - Right    v3
  END

//*************************************************************************  v3
// menu bar for main window - used by IMenuBar Class                  *   .
//  Text String <-> Menu Item ID (Command ID)                         *   .
//*************************************************************************  v3
MENU WND_MAIN                            //Main Window Menu (WND_MAIN) v3
  BEGIN
    SUBMENU "~Alignment", MI_ALIGNMENT       //Alignment Submenu         v3
      BEGIN
        MENUITEM "~Left",   MI_LEFT          //Left Menu Item           v3
        MENUITEM "~Center", MI_CENTER         //Center Menu Item          v3
        MENUITEM "~Right",  MI_RIGHT         //Right Menu Item          v3
      END
  END
```

Hello World — Version 3

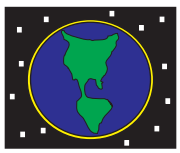
The resource file for Version 3 contains two primary additions. The first is the text strings that are assigned to the new string constants that were defined in AHELLOW3.H. These text strings are used in the status area to show the state of the static “Hello, World!!!” text string in the client area. 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 will appear on the menu bar (Alignment) and pull-down menu (Left, Center, and Right) to indicate the choices that will be available. Each text string is assigned to a constant, also defined in AHELLOW3.H.

The tilde (~) to the left of the first letter in each text string indicates that those letters can be used in combination with the Alt key to provide shortcut keys for the application. For example, pressing Alt-R will align the “Hello, World!!!” text string on the right side of the main window, just as if the Right choice had been selected from the pull-down menu.

The Icon File

AHELLOW3.ICO is used as both the title bar icon and the icon that is displayed when the application is minimized. We cannot provide a listing for the AHELLOW3.ICO file, but this is how the icon appears when minimized:



Hello World Icon

Figure 45. Hello World Icon

AHELLOW3.DEF

The AHELLOW3.DEF file is required for the same reasons that AHELLOW1.DEF was needed for Version 1. See “The Module Definition File” on page 148 if you need to review the reasons for creating a .DEF file.

The only difference between the two .DEF files used in Version 1 and Version 3 is the change in the version number.

```
NAME    HELLO3    WINDOWAPI

DESCRIPTION 'Hello World Sample C++ Program - Version 3'

CODE    LOADONCALL MOVEABLE
DATA    MOVEABLE   MULTIPLE

HEAPSIZE  8192
STACKSIZE 256
```

Tasks Performed by Version 3

The following sections describe each of the tasks performed by Version 3 of the Hello World application that were not described already for Version 2. Those tasks are:

- Creating a status area using a static text control
- Putting text in a static text control for a status line
- Specifying the location and height of the status area
- Setting `AHelloWindow` as the event handler
- Creating a menu bar
- Setting an initial check mark in the pull-down menu
- Adding command processing to set the static text control alignment

Tasks this version performs that were described for Version 2 are:

- Creating the main window
- Getting the current application and running it
- Constructing the main window, which involves the following:
  - Creating a static text control
  - Setting a text string from a resource file
  - Putting a text string into a static text control
  - Aligning the text

- Setting the static text control in the main window
- Setting the window title and title bar icon from a resource file
- Creating and setting the information area below the client area
- Setting the focus to the main window and showing the main window

Creating a Status Area Using a Static Text Control

The `IStaticText` class is used to create the static text control for displaying a text string in a status area. As shown in the following line of code, *statusLine* is declared in the `AHelloWindow` class declaration in the header file (AHELLOW3.HPP):

```
// <in AHELLOW3.HPP>
  IStaticText   * statusLine;        //Status Line at top of client window
```

In the AHELLOW3.CPP file, *statusLine* is set equal to the `IStaticText` library class to create a static text control for the status area and pass in the main window, this instance of the `AHelloWindow` class, as the parent and owner of this control:

```
// <in AHELLOW3.CPP>
statusLine=new IStaticText          //Create Status Area using Static Text v3
  (WND_STATUS, this, this);          //  Window ID, Parent, Owner Parameters.
```

The WND_STATUS constant provides the window ID for this static text control. This constant is defined in AHELLOW3.H.

Putting Text in a Static Text Control for a Status Line

The resource file (AHELLOW3.RC) comes into play now because the status area text strings are specified in the resource file:

```
// <in AHELLOW3.RC>
  STR_CENTER, "Center Alignment"           //Status Line Text - Center
  STR_LEFT,   "Left Alignment"            //Status Line Text - Left
  STR_RIGHT,  "Right Alignment"            //Status Line Text - Right
```

The following code in the .CPP file is used to get the “Center Alignment” text string from the resource file and center it in the static text control for the status area:

```
// <in AHELLOW3.CPP>
statusLine->setText(STR_CENTER);     //Set Status Text to "Center" from Res .
```

This is done because the “Hello, World!!!” text string is center aligned, both horizontally and vertically, when the static text control that displays it is created.

Specifying the Location and Height of the Status Area

The `IFrameWindow` member function `addExtension` is used to specify where the status area will be positioned and how high it will be:

```
// <in AHELLOW3.CPP>
addExtension(statusLine,           //Add Status Line above the client    .
  IFrameWindow::aboveClient,        //  and specify the location         .
  IFont(statusLine).maxCharHeight()); //  and specify height             v3
```

The *aboveClient* argument of the `Location` enumeration specifies that the static text control used to display the status area is to be located above the client window.

The `maxCharHeight` member function returns the maximum height that the status area can be based on the current font.

Setting AHelloWindow as the Event Handler

In Version 3, the `AHelloWindow` class is derived from both the `IFrameWindow` and the `ICommandHandler` classes. This is necessary because, for the first time, this application will begin to handle events, in this case, commands that align the “Hello, World!!!” text string.

The next line of code contains the `handleEventsFor` member function of the `ICommandHandler` class. This member function is used to set the event handler for the application. In this case, the *this* argument is specified, setting this instance of the `AHelloWindow` class as the event handler for the Hello World application:

```
// <in AHELLOW3.CPP>
handleEventsFor(this);             //Set self as event handler (commands)v3
```

This member function is available because the header file, AHELLOW3.HPP, includes the ICMDHDR.HPP library header file, which contains the `ICommandHandler` class.

```
 // <in AHELLOW3.HPP>
#include <icmdhdr.hpp>        //Include ICommandEvent & ICommandHandler    v3
```

Creating a Menu Bar

Now it is time to create a menu bar. In the header file, **menuBar** is defined as an instance of the `IMenuBar` class.

```
// <in AHELLOW3.HPP>
IMenuBar     * menuBar;          //Define Menu Bar                v3
```

It is now used to create a new instance of that class in the main window:

```
// <in AHELLOW3.CPP>
menuBar=new IMenuBar(WND_MAIN, this); //Create Menu Bar for main window     .
```

The WND_MAIN specified as the first argument identifies the following menu to be used from the AHELLOW3.RC resource file:

```
// <in AHELLOW3.RC>
MENU WND_MAIN                                  //Main Window Menu (WND_MAIN) v3
  BEGIN
    SUBMENU "~Alignment", MI_ALIGNMENT         //Alignment Submenu          v3
      BEGIN
        MENUITEM "~Left",   MI_LEFT           //Left Menu Item           v3
        MENUITEM "~Center", MI_CENTER           //Center Menu Item          v3
        MENUITEM "~Right",  MI_RIGHT           //Right Menu Item          v3
      END
  END
```

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:

```
// <in AHELLOW3.RC>
MI_ALIGNMENT,"Alignment Menu"            //InfoArea - Alignment Menu   v3
MI_CENTER,  "Set Center Alignment"        //InfoArea - Center Menu      .
MI_LEFT,    "Set Left Alignment"         //InfoArea - Left Menu       .
MI_RIGHT,   "Set Right Alignment"         //InfoArea - Right Menu      v3
```

This means that as the selection cursor passes over each menu item, the text string associated with that menu item will be displayed 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" will appear in the information area.

Setting an Initial Check Mark in the Pull-down Menu

The pull-down menu that is displayed when the Alignment choice is selected on the menu bar contains three choices for aligning the “Hello, World!!!” text string: Left, Center, and Right. Since this text string is aligned in the center of the client area when the application is created, a check mark should be displayed next to the Center choice the first time the pull-down menu is displayed.

The `checkItem` member function of the `IMenuBar` class allows you to place a check mark on a pull-down menu choice. The line of code below is used to place a check mark on the Center choice:

```
// <in AHELLOW3.CPP>
menuBar->checkItem(MI_CENTER);       //Place Check on Center Menu Item    v3
```

The MI_CENTER constant was defined in the AHELLOW3.RC resource file as the “Center” text string for the menu. This is not to be confused with the MI_CENTER menu item attribute defined in the string table, which is used only by the information area.

Adding Command Processing to Align the Static Text

Now we will show you how to associate commands with the menu items to align the text string.

An example of the command processing for one of the menu items follows. This code is used to left-align the “Hello, World!!!” text string in the client window:

```
// <in AHELLOW3.CPP>
  case MI_LEFT:                    //Code to Process Left Command Item   v3
    hello->setAlignment(            //Set alignment of hello text to      .
      IStaticText::centerLeft);      //  center-vertical, left-horizontal   .
    statusLine->setText(STR_LEFT);    //Set Status Text to "Left" from Res   .
    menuBar->uncheckItem(MI_CENTER);  //Uncheck Center Menu Item           .
    menuBar->checkItem(MI_LEFT);      //Place Check on Left Menu Item       .
    menuBar->uncheckItem(MI_RIGHT);   //Uncheck Right Menu Item            .
    return(true);                    //Return command processed          .
    break;                          //                              v3
```

This code does the following:

- Uses the `setAlignment` 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` function to Remove any existing check marks from the Center and Right menu items
- Uses the `checkItem` function to set a check mark on the Left item
- Returns true and quits

Compiling and Linking Version 3

Figure 46 shows the files that were used to create Version 3 of the Hello World application, their relationship to each other, and the order in which they are compiled and linked. File names are shown in uppercase letters; program names are shown in lowercase letters.

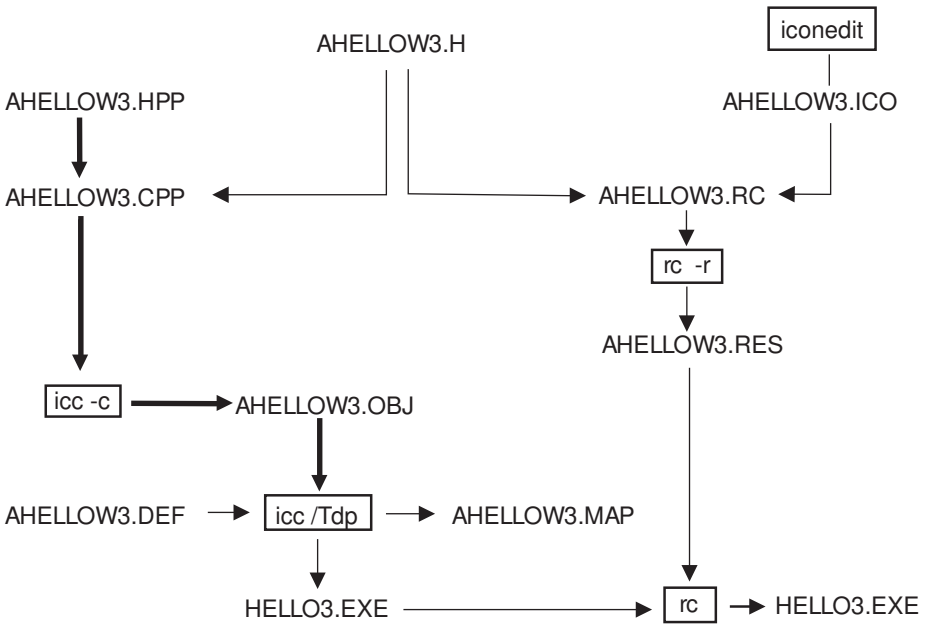


Figure 46. Compiling and Linking Version 3 of the Hello World Application

Hello World — Version 3

Chapter 14. Simple Dialogs and Push Buttons

Version 4 modifies menu bar and the pull-down menu in several significant ways. It puts an Edit choice on the menu bar and moves the Alignment choice from the menu bar to the pull-down menu. The menu items associated with the Alignment choice (Left, Center, and Right) are moved from the pull-down menu into a cascaded menu that is displayed when the Alignment choice is selected. These items are still used to align the “Hello, World!!!!” text string in the client window. However, the commands assigned to these menu items are also assigned to function keys so the keyboard can be used to bypass the menu entirely for text alignment purposes.

The Alignment choice is joined on the pull-down menu by a new Text... choice. Selecting this choice displays a dialog box that contains an entry field in which the “Hello, World!!!!” text string can be edited.

In covering these topics, this section shows you how to:

- Add a cascaded menu to a pull-down menu
- Assign command processing (event handling) to function keys
- Create a dialog box that contains an entry field
- Set the “Hello, World!!!!” text string in the entry field
- Edit the text string
- Adding push buttons and a set canvas to change the alignment

The window for Version 4 of the Hello World application looks like this:

© Copyright IBM Corp. 1993

Hello World — Version 4

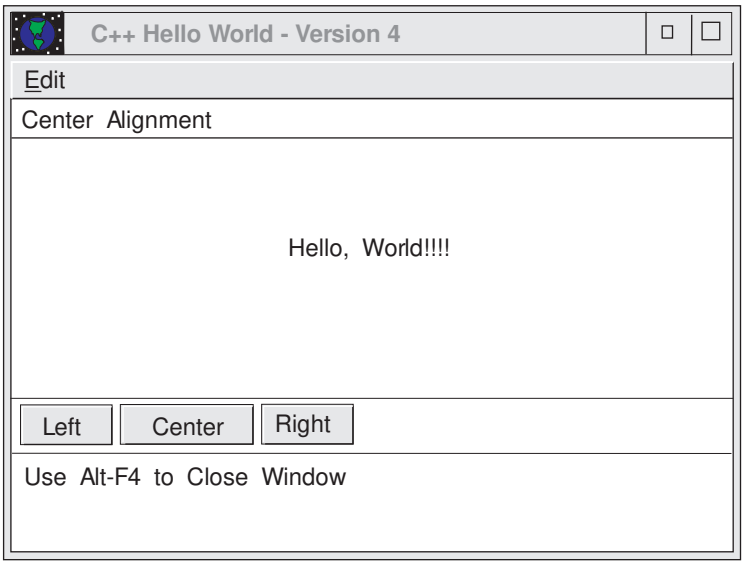


Figure 47. Version 4 of Hello World Application

Version 4 Window-Parent Relationship Diagram

Figure 48 on page 191 shows the relationship between the objects built for Version 4 of the Hello World application:

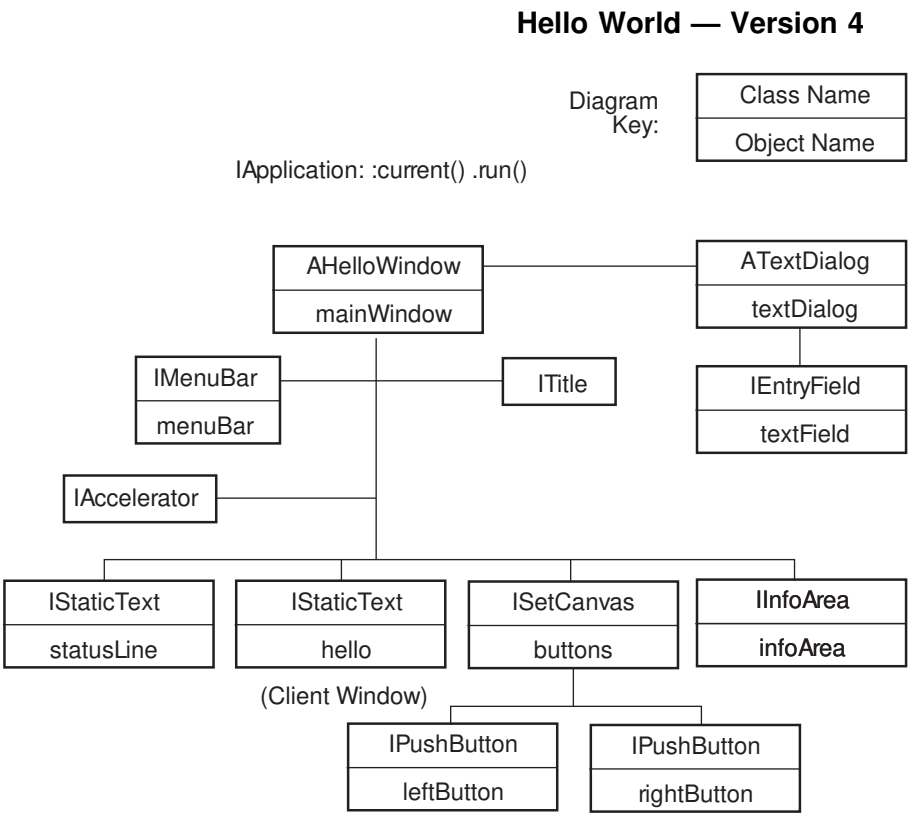


Figure 48. Window Parent Relationship Diagram

Version 4 Files

The following files contain the code used to create this window:

AHELLOW4.CPP Source code for the main procedure, main window constructor, and command processing.

AHELLOW4.HPP Header file for the `AHellowWindow` class.

AHELLOW4.H Constants definition file for HELLO4.EXE.

ADIALOG4.CPP Source code to create the `ATextDialog` class.

ADIALOG4.HPP Header file for the `ATextDialog` class.

AHELLOW4.RC Resource file for HELLO4.EXE.

AHELLOW4.ICO Icon file for HELLO4.EXE.

ADIALOG4.DLG Dialog resource source file for HELLO4.EXE.

ADIALOG4.RES Dialog resource file for HELLO4.EXE.

AHELLOW4.DEF Module definition file for HELLO4.EXE.

The Primary Source Code File

The AHELLOW4.CPP file contains the source code for the main procedure, window constructor, and menu commands. The tasks performed by this code are described in “Tasks Performed by Version 4” on page 205 and its related sections.

Here is a listing of the primary source code:

```
                                //Include IBM UI class headers:
#include <iapp.hpp>                 //IApplication Class
#include <istattxt.hpp>              //IStaticText Class
#include <iinfoa.hpp>               //IInfoArea Class                v2
#include <imenubar.hpp>              //IMenuBar Class                v3
#include <ifont.hpp>                //IFont                        v3
#include <istring.hpp>               //IString Class                  v4
#include <isetcv.hpp>               //ISetCanvas Class               v4
#include <ipushbut.hpp>              //IPushButton Class              v4

#include "ahellow4.hpp"              //Include AHelloWindow Class headers  v2
#include "ahellow4.h"               //Include our Symbolic definitions   v2
#include "adialog4.hpp"              //ATextDialog Class              v4

//*************************************************************************
// main  - Application entry point                                        *
//*************************************************************************
void main()                       //Main Procedure with no parameters
{
  AHelloWindow mainWindow (WND_MAIN);   //Create our main window on the
                                // desktop
  IApplication::current().run();       //Get the current application and
                                // run it
} /* end main */
```

Hello World — Version 4

```
//*************************************************************************
// AHelloWindow :: AHelloWindow - Constructor for our main window        *
//*************************************************************************
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow (                  //Call IFrameWindow constructor      v2
    IFrameWindow::defaultStyle()       //  Use default plus               v2
    | IFrameWindow::minimizedIcon      //  Get Minimized Icon from  RC file  v2
    | IFrameWindow::accelerator,       //  Get Accelerator Table from RC file v4
    windowId)                       //  Main Window ID
{
  hello=new IStaticText(WND_HELLO,     //Create Static Text Control
    this, this);                    //  Pass in myself as owner & parent
  hello->setText(STR_HELLO);           //Set text in Static Text Control    v2
  hello->setAlignment(               //Set Alignment to Center in both
    IStaticText::centerCenter);        //    directions
  setClient(hello);                  //Set hello control as Client Window

  infoArea=new IInfoArea(this);        //Create the information area       v2
  infoArea->setInactiveText(STR_INFO);  //Set information area text from RC   v2

  statusLine=new IStaticText          //Create Status Area using Static Text v3
    (WND_STATUS, this, this);         //  Window ID, Parent, Owner Parameters.
  statusLine->setText(STR_CENTER);     //Set Status Text to "Center" from Res .
  addExtension(statusLine,           //Add Status Line above the client    .
    IFrameWindow::aboveClient,         //  and specify the height           .
    IFont(statusLine).maxCharHeight()); //  and specify height              v3

  handleEventsFor(this);              //Set self as event handler (commands)v3
  menuBar=new IMenuBar(WND_MAIN, this); //Create Menu Bar for main window     .
  menuBar->checkItem(MI_CENTER);       //Place Check on Center Menu Item    v3

  setupButtons();                    //Setup Buttons                    v4

  sizeTo(ISize(4  ,3  ));             //Set the size of main window       v2
  setFocus();                       //Set focus to main window
  show();                           //Set to show main window

} /* end AHelloWindow :: AHelloWindow(...) */
```

Hello World — Version 4

```
//*************************************************************************  v4
// AHelloWindow :: setupButtons                                      *   .
//   Setup Buttons                                                *   .
//*************************************************************************   .
Boolean AHelloWindow :: setupButtons()  //Setup Buttons                    .
{                                 //                                .
  ISetCanvas   * buttons;            //Define canvas of buttons          .
                                  //                                .
  buttons=new ISetCanvas(WND_BUTTONS,   //Create a Set Canvas for Buttons     .
    this, this) ;                   //  Parent and Owner=me             .
  buttons->setMargin(ISize());         //Set Canvas Margins to zero         .
  buttons->setPad(ISize());           //Set Button Canvas Pad to zero       .
  leftButton=new IPushButton(MI_LEFT,   //Create Left Push Button           .
    buttons, buttons, IRectangle(),     //  Parent, Owner=Button Canvas       .
    IPushButton::defaultStyle() |       //  Use Default Styles plus          .
    IControl::tabStop);               //  tabStop                       .
  leftButton->setText(STR_LEFTB);      //Set Left Button Text              .
  centerButton=new IPushButton(MI_CENTER,//Create Left Push Button           .
    buttons, buttons, IRectangle(),     //  Parent, Owner=Button Canvas       .
    IPushButton::defaultStyle() |       //  Use Default Styles plus          .
    IControl::tabStop);               //  tabStop                       .
  centerButton->setText(STR_CENTERB);   //Set Center Button Text            .
  rightButton=new IPushButton(MI_RIGHT, //Create Right Push Button           .
    buttons, buttons, IRectangle(),     //  Parent, Owner=Button Canvas       .
    IPushButton::defaultStyle() |       //  Use Default Styles plus          .
    IControl::tabStop);               //  tabStop                       .
  rightButton->setText(STR_RIGHTB);     //Set Right Button Text             .
  addExtension(buttons,               //Add Buttons Canvas               .
    IFrameWindow::belowClient,         //  below client and               .
    3 UL);                          //  specify height in pixels         .
  return true;                      //Return                          .
} /* end AHelloWindow :: setupButtons() */                            //v4
```

Hello World — Version 4

```
//*************************************************************************  v3
// AHelloWindow :: command                                          *   .
//   Handle menu commands                                          *   .
//*************************************************************************   .
Boolean AHelloWindow :: command(ICommandEvent & cmdEvent)                 // .
{                                                                  //v3
  IString temp;                      //String to pass in/out from dialog   v4
  unsigned short value;               //Return value from dialog          v4
  switch (cmdEvent.commandId()) {      //Get command id                   v3

    case MI_CENTER:                  //Code to Process Center Command Item v3
      hello->setAlignment(            //Set alignment of hello text to      .
        IStaticText::centerCenter);    //  center-vertical, center-horizontal .
      statusLine->setText(STR_CENTER);  //Set Status Text to "Center" from Res .
      menuBar->checkItem(MI_CENTER);    //Place Check on Center Menu Item     .
      menuBar->uncheckItem(MI_LEFT);    //Uncheck Left Menu Item            .
      menuBar->uncheckItem(MI_RIGHT);   //Uncheck Right Menu Item           .
      return(true);                  //Return command processed          .
      break;                        //                              v3

    case MI_LEFT:                    //Code to Process Left Command Item   v3
      hello->setAlignment(            //Set alignment of hello text to      .
        IStaticText::centerLeft);      //  center-vertical, left-horizontal   .
      statusLine->setText(STR_LEFT);    //Set Status Text to "Left" from Res   .
      menuBar->uncheckItem(MI_CENTER);  //Uncheck Center Menu Item          .
      menuBar->checkItem(MI_LEFT);      //Place Check on Left Menu Item       .
      menuBar->uncheckItem(MI_RIGHT);   //Uncheck Right Menu Item           .
      return(true);                  //Return command processed          .
      break;                        //                              v3
```

Hello World — Version 4

```
    case MI_RIGHT:                  //Code to Process Right Command Item  v3
      hello->setAlignment(           //Set alignment of hello text to      .
        IStaticText::centerRight);     //  center-vertical, right-horizontal  .
      statusLine->setText(STR_RIGHT);   //Set Status Text to "Right" from Res  .
      menuBar->uncheckItem(MI_CENTER);  //Uncheck Center Menu Item           .
      menuBar->uncheckItem(MI_LEFT);    //Uncheck Left Menu Item             .
      menuBar->checkItem(MI_RIGHT);     //Place Check on Right Menu Item     .
      return(true);                  //Return command processed           .
      break;                        //                              v3

    case MI_TEXT:                   //Code to Process Text Command      v4
      {
      temp=hello->text();            //Get current Hello text             .
      infoArea->setInactiveText(      //Set Info Area to Dialog Active     .
        STR_INFODLG);                //  Text from Resource File          .
      ATextDialog * textDialog=new     //Create a Text Dialog              .
        ATextDialog(temp, this);       //                                  .
      textDialog->showModally();       //Show this Text Dialog as Modal     .
      value=textDialog->result();      //Get result (eg OK or Cancel)       .
      if (value != DID_CANCEL)         //Set new string if not canceled     .
        hello->setText(temp);          //Set Hello to Text from Dialog      .
      infoArea->setText(STR_INFO);     //Set Info Text to "Normal" from Res   .
      delete textDialog;              //Delete textDialog                  .
      return(true);                  //Return Command Processed           .
      break;                        //                              v4
      }

  } /* end switch */                 //                                v3

  return(false);                    //Return command not processed      v3
} /* end HelloWindow :: command(...) */                                 //v3
```

The AHelloWindow Class Header File

Like AHELLOW3.HPP, AHELLOW4..HPP contains the class definition and interface specifications for the `AHelloWindow` class, with a few modifications for Version 4. Here is the source listing for AHELLOW4.HPP:

```
#ifndef AHELLOWINDOW_HPP
#define AHELLOWINDOW_HPP

#include <iframe.hpp>         //Include IFrameWindow Class Header
#include <icmdhdr.hpp>        //Include ICommandEvent & ICommandHandler    v3

class ATextDialog;           //Define the ATextDialog Class             v4

//*************************************************************************
// Class:  AHelloWindow                                              *
//                                                                 *
// Purpose: Main Window for C++ Hello World sample application          *
//        It is a subclass of IFrameWindow & ICommandHandler          *  v3
//                                                                 *
//*************************************************************************
class AHelloWindow : public IFrameWindow,
                 public ICommandHandler                          //v3
{
 public:                           //Define the Public Information
  AHelloWindow(unsigned long windowId); //Constructor for this class

 protected:                         //Define Protected Member          v3
  Boolean command(ICommandEvent& cmdEvent);                          //v3
  virtual Boolean setupButtons();    //Setup Buttons                    v4

 private:                          //Define Private Information
  IStaticText   * hello;           //Hello contains "Hello, World" text
  IInfoArea    * infoArea;         //Define an Information Area        v2
                                 //  Control to create an information  .
                                 //  area beneath the client area     v2
  IStaticText   * statusLine;       //Status Line at top of client window v3
  IMenuBar     * menuBar;          //Define Menu Bar                  v3
  ATextDialog   * textDialog;       //Define Text Dialog               v4
  IPushButton   * leftButton;        //Define Left Button               .
  IPushButton   * centerButton;       //Define Center Button            .
  IPushButton   * rightButton;        //Define Right Button             v4
};
#endif
```

The Constants Definition File

AHELLOW4.H contains the constant definitions for this application. These constants and their definitions, which provide the IDs for the application window components, are shown in the following code:

```
#ifndef AHELLOWINDOW_H
#define AHELLOWINDOW_H

//*************************************************************************
// window ids - used by IWindow constructors (eg IStaticText, AHelloWindow)*
//*************************************************************************
#define WND_MAIN         x1           //Main Window ID

#define WND_HELLO        x1 1         //Hello World Window ID
#define WND_INFO         x1 12        //Information Area                v2
#define WND_STATUS       x1 11        //Status Line Window ID            v3
#define WND_TEXTDIALOG    x1 13        //Text Dialog Window ID            v4
#define WND_BUTTONS      x1 21        //Button Canvas Window ID          v4

//*************************************************************************  v2
// string ids - used to relate resources to IStaticText and ITitle        *   .
//*************************************************************************  v2
#define STR_HELLO        x12          //Hello World String ID            v2
#define STR_INFO         x122         //Info String ID                 v2
#define STR_INFODLG      x1221        //Info String ID                  v4
#define STR_CENTER       x123         //Center Alignment Status String ID  v3
#define STR_LEFT         x1231        //Left Alignment Status String ID     .
#define STR_RIGHT        x1232        //Right Alignment Status String ID   v3
#define STR_CENTERB      x124         //Center Button String ID          v4
#define STR_LEFTB        x1241        //Left Button String ID            .
#define STR_RIGHTB       x1242        //Right Button String ID           v4
```

```
//*************************************************************************  v3
// menu ids - used on relate command ID to Menu Items and Function Keys   *  .
//*************************************************************************   .
#define MI_ALIGNMENT     x15         //Alignment Menu ID               .
#define MI_CENTER        x15 1       //Center Menu ID                   .
#define MI_LEFT          x15 2       //Left Menu ID                    .
#define MI_RIGHT         x15 3       //Right Menu ID                  v3
#define MI_EDIT          x15 4       //Edit Menu ID                   v4
#define MI_TEXT          x15 5       //Text Menu ID                   v4
#define MI_HELP          x151        //Help Menu ID                   v4

//*************************************************************************  v4
// dialog ids - used to relate dialog fields to controls/commands        *  .
//*************************************************************************   .
#ifndef DID_OK                      //                                .
#define DID_OK          x   1        //OK Button (Defined by OS/2)        .
#endif                             //                                 .
#ifndef DID_CANCEL                  //                                  .
#define DID_CANCEL       x   2       //Cancel Button (Defined by OS/2)     .
#endif                             //                                 .
#define DID_ENTRY        x16 3       //Dialog Entry Field ID             .
#define DID_STATIC       x16 4       //Dialog Static Text               v4

#endif
```

For Version 4, the constants definition file contains new window IDs (WND_TEXTDIALOG and WND_BUTTONS) for the text dialog and the push button controls on the canvas, respectively. It also contains new string IDs (STR_CENTERB, STR_LEFTB, and STR_RIGHTB) for the text strings used in the push buttons. In addition, menu IDs (MI_EDIT, MI_HELP, and MI_TEXT) have been added for the menu bar Edit and Help choices and the Text choice in the pull-down menu.

The Text Dialog Source Code File

The ADIALOG4.CPP file contains the source code for the text dialog window constructor, the `ATextDialog` class, created for Version 4.

Here is a listing of the source code:

```
//*************************************************************************
// The entire file was created at version 4                              *
//*************************************************************************

#include <ientryfd.hpp>             //IEntryField Class
#include <icmdevt.hpp>              //ICommandEvent
#include <istring.hpp>              //IString Class
#include <ireslib.hpp>              //IResourceLibrary/IResourceId Class
```

Hello World — Version 4

```
#include "ahellow4.h"               //Include our Symbolic definitions
#include "adialog4.hpp"             //ATextDialog Class

//*************************************************************************
// ATextDialog :: ATextDialog - Constructor for text dialog window       *
//*************************************************************************
ATextDialog :: ATextDialog(IString & textString, IWindow * parent)
          : IFrameWindow(IResourceId(WND_TEXTDIALOG), ownerWnd),
             textValue(textString)
{
  ICommandHandler::handleEventsFor(this);//Set self as command event handler

  textValue=textString ;             //Save textValue for exit of dialog
  textField=new IEntryField(DID_ENTRY,  //Create entry field object using dialog
    this);                          //  entry field
  textField->setText(textString);      //Set top current "Hello, World" text
  textField->setFocus();             //Set focus to entry field

} /* end ATextDialog :: ATextDialog(...) */

//*************************************************************************
// ATextDialog :: ~ATextDialog - Destructor                               *
//*************************************************************************
ATextDialog :: ~ATextDialog()
{
} /* end ATextDialog :: ~TextDialog(...) */

//*************************************************************************
// ATextDialog :: command - Process Commands                              *
//*************************************************************************
Boolean ATextDialog :: command(ICommandEvent& cmdevt)
{
  switch(cmdevt.commandId()) {
    case DID_OK: // DID_OK           //Process OK Button
      textValue=textField->text();     //Get Text from Dialog Entry Field
      dismissModal(DID_OK);          //Dismiss Dialog - Allow focus to main
      return(true);                 //Return Processing Completed
      break;

    case DID_CANCEL: // DID_CANCEL     //Process CANCEL Button
      dismissModal(DID_CANCEL);       //Dismiss Dialog - Allow focus to main
      return(true);                 //Return Processing Completed
      break;
  }/* end switch */

  return(false);                   //Allow Default Processing to occur
} /* end ATextDialog :: command(...) */
```

The ATextDialog Class Header File

The ADIALOG4.HPP contains the class definition and interface specifications for the `ATextDialog` class. Here is the source listing for ADIALOG4.HPP:

```
#ifndef ATEXTDIALOG_HPP
#define ATEXTDIALOG_HPP

//*************************************************************************
// The entire file was created at version 4                              *
//*************************************************************************

#include <iframe.hpp>               //IFrameWindow Class (Parent)
#include <icmdhdr.hpp>              //ICommandHandler (Parent)

class IEntryField;

//*************************************************************************
// Class:  ATextDialog                                                  *
//                                                                      *
// Purpose: Dialog window for the C++ Hello World sample application.     *
//        It is a subclass of IFrameWindow, ICommandHandler             *
//                                                                      *
//*************************************************************************
class ATextDialog : public IFrameWindow, public ICommandHandler
{
public:
  ATextDialog (IString & textString, IWindow * ownerWnd) ;
  ~ATextDialog();

protected:
  virtual Boolean
    command(ICommandEvent& cmdevt);    //Process the dialog command events

private:
  IEntryField * textField ;          //Entry Field to Edit Hello Text
  IString & textValue ;              //String Value for in/out of dialog

}; // TextDialog

#endif
```

The Resource File

Version 4 of the Hello World application provides a resource file, AHELLOW4.RC. This resource file associates an icon and several text strings with the symbols defined in the AHELLOW4.H file shown in “The Constants Definition File” on page 198. It also contains resources for the menu bar, the accelerator keys (CUA calls these “shortcut” keys), and the text dialog.

AHELLOW4.H is included in this resource file so the icon, text strings, and other resources can be associated with the appropriate IDs. OS.H is included because it is the top level include file that includes all the files necessary for writing an OS/2 application.

Here is the code used in the AHELLOW4.RC file:

```
#include <os2.h>                          //Include os2.h            v4
#include "ahellow4.h"                      //Symbolic Definitions       v2

//*************************************************************************  v2
// icon and bitmap resources                                        *   .
//  Symbolic Name (ID) <-> icon filename                              *   .
//*************************************************************************   .
ICON WND_MAIN ahellow4.ico                 //Title Bar Icon (use same id)v2

//*************************************************************************  v2
// string resources - used by IStaticText & ITitle Classes              *   .
//  Symbolic Name (ID) <-> Text String                               *   .
//*************************************************************************  v2
STRINGTABLE
  BEGIN
   STR_HELLO,  "Hello, World!!!!"           //Hello World String        v4
   WND_MAIN,   "C++ Hello World - Version 4"   //Title Bar String (main id)  v4
   STR_INFO,   "Use Alt-F4 to Close Window"   //Information Area String    v2
   MI_EDIT,    "Edit Menu"                 //InfoArea - Edit Menu       v4    .
   MI_ALIGNMENT,"Alignment Menu"            //InfoArea - Alignment Menu  v3
   MI_CENTER,  "Set Center Alignment"        //InfoArea - Center Menu      .
   MI_LEFT,    "Set Left Alignment"         //InfoArea - Left Menu        .
   MI_RIGHT,   "Set Right Alignment"         //InfoArea - Right Menu      v3
   MI_TEXT,    "Display Edit Dialog"         //InfoArea - Text Menu       v4
   STR_INFODLG,"Modal Edit Text Dialog Active" //Information Area String    v4
   STR_CENTER, "Center Alignment"           //Status Line Text - Center  v3
   STR_LEFT,   "Left Alignment"             //Status Line Text - Left     .
   STR_RIGHT,  "Right Alignment"            //Status Line Text - Right   v3
   STR_LEFTB,  "Left"                      //String for Left Button     v4
   STR_CENTERB,"Center"                     //String for Center Button    .
   STR_RIGHTB, "Right"                     //String for Right Button    v4
  END
```

Hello World — Version 4

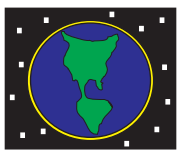
```
//*************************************************************************  v3
// menu bar for main window - used by IMenuBar Class                    *   .
//  Text String <-> Menu Item ID (Command ID)                          *   .
//*************************************************************************  v3
MENU WND_MAIN                              //Main Window Menu (WND_MAIN) v3
  BEGIN
   SUBMENU "~Edit", MI_EDIT                  //Edit Submenu             v4
     BEGIN
      SUBMENU "~Alignment", MI_ALIGNMENT     //Alignment Submenu         v3
        BEGIN
         MENUITEM "~Left\tF7",   MI_LEFT    //Left Menu Item - F7 Key    v4
         MENUITEM "~Center\tF8", MI_CENTER  //Center Menu Item - F8 Key   v4
         MENUITEM "~Right\tF9",  MI_RIGHT   //Right Menu Item - F9 Key    v4
        END
      MENUITEM "~Text...", MI_TEXT          //Text Menu Item            v4
     END
  END

//*************************************************************************  v4
// Accelerator (key) table resources - used by IAccelerator Class        *   .
//  Key Value <-> Menu Item ID (Command ID)                            *   .
//*************************************************************************   .
ACCELTABLE WND_MAIN                          //Acc. Table for Main Window   .
  BEGIN                                   //                          .
   VK_F7,  MI_LEFT,   VIRTUALKEY            //F7 - Left Command          .
   VK_F8,  MI_CENTER, VIRTUALKEY             //F8 - Center Command        .
   VK_F9,  MI_RIGHT,  VIRTUALKEY             //F9 - Right Command         .
  END                                     //                       v4

//*************************************************************************  v4
// dialog resources - used by ATextDialog Class                          *   .
//*************************************************************************   .
rcinclude adialog4.dlg                      //Text Dialog Template       v4
```

The resource file for Version 4 contains two primary additions. The first is the accelerator table of text strings that are assigned to the function keys. These text strings 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, pressing the F7 key is the same as selecting the Left choice in the cascaded menu.

The second addition is an `rcinclude` statement that includes the text dialog template. See “The Text Dialog Template” on page 204 for information about that file.

The Icon File

AHELLOW4.ICO is used as both the title bar icon and the icon that is displayed when the application is minimized. We cannot provide a listing for the AHELLOW4.ICO file, but this is how the icon appears when minimized:



Hello World Icon

Figure 49. Hello World Icon

The Text Dialog Template

ADIALOG4.DLG contains the template used to build the text dialog. Here is that template:

```
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, 4 , 14
      PUSHBUTTON     "Cancel", DID_CANCEL, 49, 4, 4 , 14
      LTEXT         "Edit Text:", DID_STATIC, 8, 62, 69, 8
      ENTRYFIELD     "", DID_ENTRY, 8, 44, 114, 8, ES_MARGIN
   END
END
```

The Text Dialog Resource File

ADIALOG4.RES is created by the resource compiler as input to HELLO4.EXE.

AHELLOW4.DEF

The AHELLOW4.DEF file is required for the same reasons that AHELLOW1.DEF was needed for Version 1. See “The Module Definition File” on page 148 if you need to review the reasons for creating a .DEF file.

The only difference between the two .DEF files used in Version 1 and Version 4 is the change in the version number.

```
NAME    HELLO4    WINDOWAPI

DESCRIPTION 'Hello World Sample C++ Program - Version 4'

CODE    LOADONCALL MOVEABLE
DATA    MOVEABLE   MULTIPLE

HEAPSIZE  8192
STACKSIZE 256
```

Tasks Performed by Version 4

The following sections describe each of the tasks performed by Version 4 of the Hello World application that were not described already for previous versions. Those tasks are:

- Adding a cascaded menu to a pull-down menu
- Adding accelerator, or shortcut, keys for the Left (F7), Center(F8), and Right(F9) cascaded menu choices
- Adding the `ADialogText` class to allow the user to edit the “Hello, World!!!!” text string
- Adding push buttons in a set canvas

Tasks this version performs that were described for Version 3 are:

- Creating a status area using a static text control
- Putting text in a static text control for a status line
- Specifying the location and height of the status area
- Setting `AHelloWindow` as the event handler
- Creating a menu bar
- Setting an initial check mark in the pull-down menu
- Adding command processing to set the static text control alignment

Hello World — Version 4

Tasks this version performs that were described for Versions 1 and 2 are:

- Creating the main window
- Getting the current application and running it
- Constructing the main window, which involves the following:
  - Creating a static text control
  - Setting a text string from a resource file
  - Putting a text string into a static text control
  - Aligning the text
  - Setting the static text control in the main window
  - Setting the window title and title bar icon from a resource file
  - Creating and setting the information area below the client area
  - Setting the focus to the main window and showing the main window

Modifying the Menu Bar and Pull-down Menu

For Version 4, we made several modifications to the menu bar and its associated pull-down menu. Following is the code from the AHELLOW4.RC file used to define this menu:

```
// <in AHELLOW4.RC>
MENU WND_MAIN                              //Main Window Menu (WND_MAIN) v3
  BEGIN
    SUBMENU "~Edit", MI_EDIT                 //Edit Submenu             v4
      BEGIN
        SUBMENU "~Alignment", MI_ALIGNMENT     //Alignment Submenu         v3
          BEGIN
            MENUITEM "~Left\tF7",   MI_LEFT    //Left Menu Item - F7 Key    v4
            MENUITEM "~Center\tF8", MI_CENTER   //Center Menu Item - F8 Key   v4
            MENUITEM "~Right\tF9",  MI_RIGHT    //Right Menu Item - F9 Key    v4
          END
        MENUITEM "~Text...", MI_TEXT          //Text Menu Item           v4
      END
  END
```

Adding a Cascaded Menu to a Pull-down Menu

First, we changed the Alignment item on the menu bar to Edit. The Alignment choice is now in the pull-down menu. Selecting this choice displays a cascaded menu, which is a menu that is displayed to the right of the pull-down menu. An arrow next to the Alignment choice indicates that a cascaded menu will be displayed when it is selected, as shown in Figure 50.

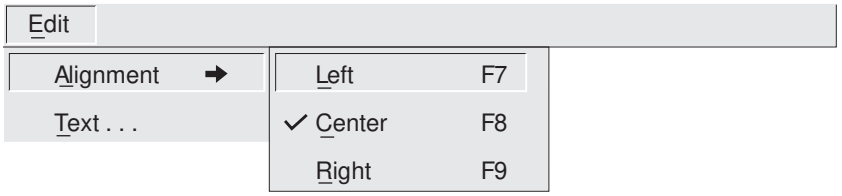


Figure 50. Cascaded Menu for Version 4 of Hello World

Adding Accelerator or Shortcut Keys to the Application

The Left, Center, and Right choices, which were items in the pull-down menu in Version 3, are now in the cascaded menu, and a function key is displayed to the right of each choice. These function keys are called accelerator, or shortcut, keys and perform the same actions as the menu items. These keys (F7, F8, and F9) can be used in place of the Left, Center, and Right menu items to align the “Hello, World!!!!” text string.

The default processing of the accelerator style is to use the accelerator that matches the frame window id. In our example, the frame window id is WND_MAIN. For Version 4, the following line of code is added to the main window constructor:

```
// <in AHELLOW4.CPP>
| IFrameWindow::accelerator,      //  Get Accelerator Table from RC file v4
```

This line of code gets the accelerator table from the resource file to define accelerator, or shortcut, keys for the Hello World application. Here is the code for the accelerator table:

Hello World — Version 4

```
// <in AHELLOW4.RC>
ACCELTABLE WND_MAIN                          //Acc. Table for Main Window   .
  BEGIN                                    //                             .
    VK_F7,  MI_LEFT,   VIRTUALKEY          //F7 - Left Command          .
    VK_F8,  MI_CENTER, VIRTUALKEY           //F8 - Center Command        .
    VK_F9,  MI_RIGHT,  VIRTUALKEY           //F9 - Right Command         .
  END                                      //                        v4
```

Adding a Dialog to a Pull-down Menu

As Figure 50 on page 207 shows, the final modification to the pull-down menu is the addition of the Text... choice. The ellipsis (...) indicates that selecting this choice causes a dialog to be displayed. In this case, the dialog that is displayed is a text dialog that uses the entry field control to allow the user to edit the “Hello, World!!!!” text string. The dialog looks like this:

Figure 51. Text Dialog for Version 4 of Hello World

Processing the Text Menu Item: The following code from the AHELLOW4.CPP file processes this menu item:

```
// <in AHELLOW4.CPP>
case MI_TEXT:                    //Code to Process Text Command      v4
  {
  temp=hello->text();             //Get current Hello text            .
  infoArea->setInactiveText(       //Set Info Area to Dialog Active     .
    STR_INFODLG);                //  Text from Resource File          .
  ATextDialog * textDialog=new     //Create a Text Dialog              .
    ATextDialog(temp, this);       //                                  .
  textDialog->showModally();       //Show this Text Dialog as Modal     .
  value=textDialog->result();      //Get result (eg OK or Cancel)       .
  if (value != DID_CANCEL)         //Set new string if not canceled     .
    hello->setText(temp);          //Set Hello to Text from Dialog      .
  infoArea->setText(STR_INFO);      //Set Info Text to "Normal" from Res  .
  delete textDialog;               //Delete textDialog                 .
  return(true);                    //Return Command Processed          .
  break;                          //                              v4
  }
```

Getting the Text String for the Dialog: The `IString` class uses the `temp` data member to get the text string for the dialog. The following code was added to the declaration of the `command` member function in AHELLOW4.CPP to accomplish this:

```
// <in AHELLOW4.CPP>
IString temp;                    //String to pass in/out from dialog   v4
```

This means the `IString` class must be included:

```
// <in AHELLOW4.CPP>
#include <istring.hpp>              //IString Class                   v4
```

The `temp` data member is set to the text string that is currently in the `hello` control, the static text control that contains the “Hello, World!!!!” text string.

```
// <in AHELLOW4.CPP>
temp=hello->text();             //Get current Hello text            .
```

Hello World — Version 4

Putting Text for the Dialog Status in the Information Area: The STR_INFODLG constant is used to put a text string in the information area to show that the dialog is active:

```
// <in AHELLOW4.CPP>
infoArea->setInactiveText(      //Set Info Area to Dialog Active     .
  STR_INFODLG);               //  Text from Resource File          .
```

This constant is defined in the AHELLOW4.RC file:

```
// <in AHELLOW4.RC>
STR_INFODLG,"Modal Edit Text Dialog Active" //Information Area String    v4
```

Creating the Text Dialog: The `textDialog` data member is used to create an instance of the `ATextDialog` class, a new class that was created as a subclass of the `IFrameWindow` class:

```
// <in AHELLOW4.CPP>
textDialog=new ATextDialog      //Create a Text Dialog             .
  (temp, this);               //  pass in temp text string, self    .
```

The `temp` data member is used to pass the current text string to the dialog.

The code for the text dialog is provided by 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.

In addition, the dialog template is contained in the ADIALOG.DLG file. The AHELLOW4.RC resource file uses the following line of code to include the dialog template:

```
// <in AHELLOW4.RC>
rcinclude adialog4.dlg                    //Text Dialog Template      v4
```

See the following for listings of the files that provide the code used to create the text dialog:

- "The Text Dialog Source Code File" on page 199
- "The ATextDialog Class Header File" on page 201
- "The Text Dialog Template" on page 204

Adding Push Buttons in a Set Canvas

The `setupButtons` function is used to set up push buttons that can be used as an alternate way to align the “Hello, World!!!!” text string. This function is declared in AHELLOW4.HPP as follows:

```
// <in AHELLOW4.HPP>
virtual Boolean setupButtons();    //Setup Buttons                  v4
```

The function is specified in AHELLOW4.CPP with no arguments, as follows:

```
// <in AHELLOW4.CPP>
setupButtons();                  //Setup Buttons                  v4
```

The following sections describe the `setupButtons` function.

Defining the setupButtons Member Function

First, `setupButtons` is defined as a member function of `AHellowWindow`:

```
// <in AHELLOW4.CPP>
Boolean AHelloWindow :: setupButtons()  //Setup Buttons                  .
```

Creating the Set Canvas

The `buttons` data member is defined as an instance of the `ISetCanvas` class to set a canvas area in which the push buttons will be positioned. See “Set Canvas” on page 47 for more information about the `ISetCanvas` features described in this chapter.

```
// <in AHELLOW4.CPP>
ISetCanvas   * buttons;          //Define canvas of buttons          .
```

To make this class available to the application, the ISETCV.HPP library header file must be included:

```
// <in AHELLOW4.CPP>
#include <isetcv.hpp>              //ISetCanvas Class               v4
```

Next, the `buttons` data member is used to create a set canvas control with the main window as the parent and owner of the control:

```
// <in AHELLOW4.CPP>
buttons=new ISetCanvas(WND_BUTTONS,   //Create a Set Canvas for Buttons     .
  this, this) ;                    //  Parent and Owner=me              .
```

The WND_BUTTONS constant provides the window ID for this set canvas control. This constant is defined in AHELLOW4.H:

```
// <in AHELLOW4.H>
#define WND_BUTTONS      x1 21       //Button Canvas Window ID          v4
```

Setting the Canvas Margin and Pad to Zero

The canvas margin and pad are set to zero:

```
// <in AHELLOW4.CPP>
  buttons->setMargin(ISize());        //Set Canvas Margins to zero         .
  buttons->setPad(ISize());          //Set Button Canvas Pad to zero      .
```

Defining the Push Buttons

The following code defines three push button data members in the header file:

```
   // <in AHELLOW4.HPP>
   IPushButton   * leftButton;         //Define Left Button              .
   IPushButton   * centerButton;       //Define Center Button             .
   IPushButton   * rightButton;        //Define Right Button            v4
```

Creating the Push Buttons

The AHELLOW4.CPP file includes the IPUSHBUT.HPP library header file, making the `IPushButton` class available to the application:

```
// <in AHELLOW4.CPP>
#include <ipushbut.hpp>              //IPushButton Class                v4
```

This allows the data members defined in the AHELLOW4.HPP file to be used to create three pushbuttons in the set canvas: Left, Center, and Right. The following code shows how this is done for the Left push button:

```
// <in AHELLOW4.CPP>
leftButton=new IPushButton(MI_LEFT,   //Create Left Push Button           .
  buttons, buttons, IRectangle(),    //  Parent, Owner=Button Canvas      .
  IPushButton::defaultStyle() |      //  Use Default Styles plus         .
  IControl::tabStop);              //  tabStop                        .
```

This code creates a new instance of a push button control, specifying that it is to use the command processing that is associated with the MI_LEFT menu item attribute to align the “Hello, World!!!!” text string on the left side of the client window. Other than the data member used (centerButton is used for the Center push button and rightButton is used for the Right push button), this attribute is the only difference in the code that is used to create all three push buttons. For the Center push button, the MI_CENTER menu item attribute is specified, while MI_RIGHT is used for the Right push button.

The set canvas control is identified as the owner and parent of the push button control.

The `defaultStyle` member function specifies that the default style defined for the `IPushButton` class is to be used for this push button with one exception. The `tabStop` style, inherited from the `IControl` class, is specified so the user can tab to this push button.

Setting Text in the Push Buttons

The `setText` function is used to set text strings in each push button. Here is the code used to set the text in the Left push button:

```
// <in AHELLOW4.CPP>
leftButton->setText(STR_LEFTB);      //Set Left Button Text              .
```

Other than the data member for which the text is being 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 used to put text in the other two push buttons is the STR_LEFTB constant, which is associated with the appropriate text string in the AHELLOW4.RC file. Here are the text string associations for all three push buttons:

```
// <in AHELLOW4.RC>
STR_LEFTB,  "Left"                    //String for Left Button     v4
STR_CENTERB,"Center"                  //String for Center Button    .
STR_RIGHTB, "Right"                   //String for Right Button    v4
```

Compiling and Linking Version 4

Figure 52 shows the files that were used to create Version 4 of the Hello World application, their relationship to each other, and the order in which they are compiled and linked. File names are shown in uppercase letters; program names are shown in lowercase letters.

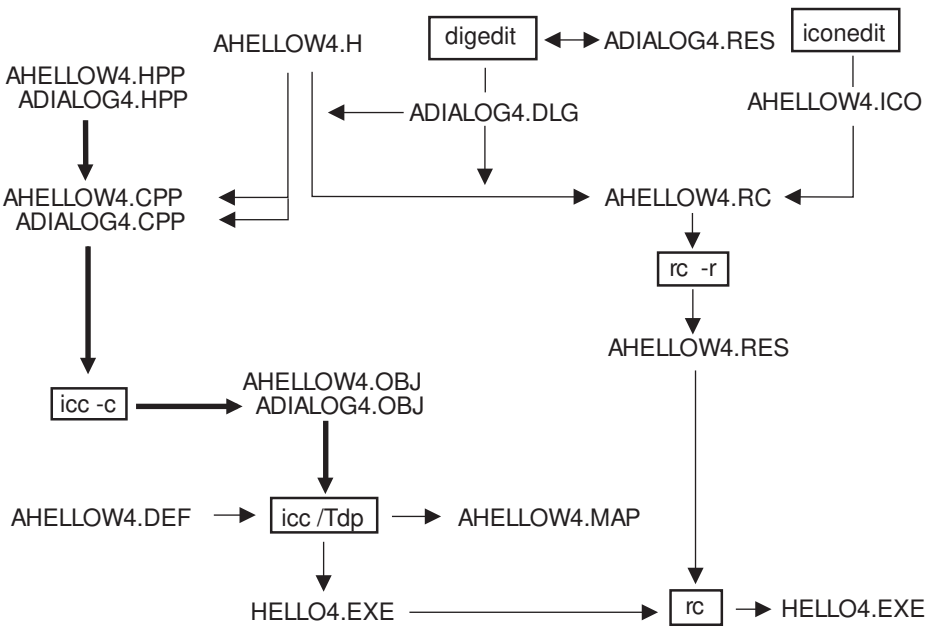


Figure 52. Compiling and Linking Version 4 of the Hello World Application

Hello World — Version 4

Chapter 15. Canvas, User-Created Control, and Help

Version 5 of the Hello World application covers the following topics:

- Coding a new control (`AEarthWindow`) using PM graphics calls
- Adding `AEarthWindow` to the bottom of the client area
- Adding help windows for the main, dialog, and entry field windows
- Using a split canvas as the client area
- Adding a list box to the client area to change the “Hello, World!!!!!” text

The window for Version 5 of the Hello World application looks like this:

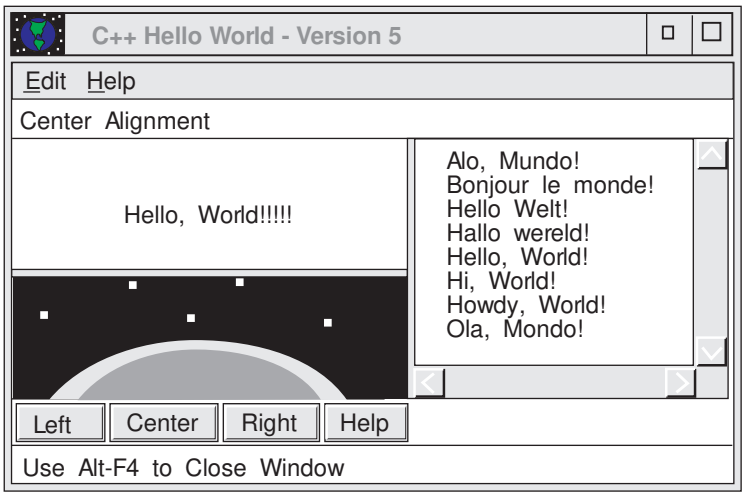


Figure 53. Version 5 of Hello World Application

© Copyright IBM Corp. 1993

Version 5 Window-Parent Relationship Diagram

Figure 54 shows the relationship between the objects built for Version 5 of the Hello World application:

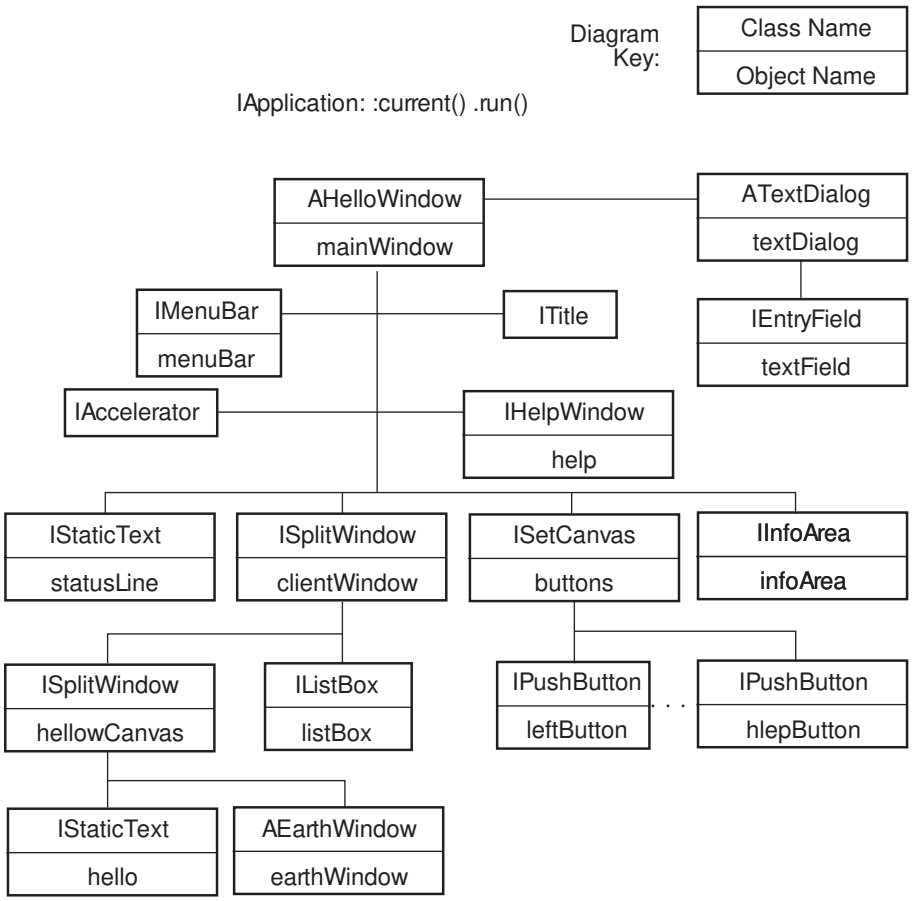


Figure 54. Window Parent Relationship Diagram

Version 5 Files

The following files contain the code used to create this window:

AHELLOW5.CPP Source code for main procedure and `AHelloWindow` class.

AHELLOW5.HPP Class header file for `AHellowWindow`.

AHELLOW5.H Symbolic defintion file for HELLO5.EXE.

ADIALOG5.CPP Source code to create the `ATextDialog` class.

ADIALOG5.HPP Class header file for `ATextDialog`.

AEARTHW5.CPP Source code to create the `AEarthWindow` class.

AEARTHW5.HPP Class header file for `AEarthWindow`.

AHELLOW5.RC Resource file for HELLO5.EXE.

AHELLOW5.ICO Icon file for HELLO5.EXE.

ADIALOG5.DLG Dialog resource source file for HELLO5.EXE.

ADIALOG5.RES Dialog resource file for HELLO5.EXE.

AHELLOW5.IPF Help file for HELLO5.EXE.

AHELLOW5.DEF Module definition file for HELLO5.EXE.

The Primary Source Code File

The AHELLOW5.CPP file contains the source code for the main procedure, window constructor, and menu commands. The tasks performed by this code are described in “Tasks Performed by Version 5” on page 240 and its related sections.

Here is a listing of the lines in the primary source code that were added or changed for Version 5:

```
⋮
#include <ihelp.hpp>                //IHelpWindow Class
#include <ihelphdr.hpp>              //IHelpHandler Class
#include <isplitcv.hpp>              //ISplitCanvas Class
#include <ilistbox.hpp>              //IListBox
⋮
#include "aearthw5.hpp"              //AEarthWindow Class
```

Hello World — Version 5

```
//*************************************************************************
// main  - Application entry point                                        *
//*************************************************************************
⋮
<Same as Version 4>
⋮
//*************************************************************************
// AHelloWindow :: AHelloWindow - Constructor for our main window        *
//*************************************************************************
⋮
<Same as Version 4>
⋮
  setupClient();                    //Setup Client Window
  setupStatusArea();                 //Setup Status Area
  setupInfoArea();                  //Setup Information Area
⋮
  setupMenuBar();                   //Setup Menu Bar
  setupHelp();                      //Setup Help
⋮

//*************************************************************************
// AHelloWindow :: setupButtons                                           *
//   Setup Buttons                                                      *
//*************************************************************************
⋮
<Same as Version 4>
⋮
  IPushButton   * helpButton;         //Define Help Button
⋮
  helpButton=new IPushButton(MI_HELP,   //Create Help Push Button
    buttons, buttons, IRectangle(),    //  Parent, Owner=Button Canvas
    IPushButton::defaultStyle() |      //  Use Default Styles plus
    IPushButton::help |              //  Help Style
    IControl::tabStop);              //  tabStop
  helpButton->setText(STR_HELPB);      //Set Help Button Text
⋮
```

Hello World — Version 5

```
//*************************************************************************
// AHelloWindow :: setupHelp()                                            *
//   Setup Help                                                         *
//*************************************************************************
Boolean AHelloWindow :: setupHelp()    //Setup Help Area
{                                     //
  help=new IHelpWindow(HELP_TABLE,     //Create Help Window Object
    this);                           //Setup Help info
  help->addLibraries("AHELLOW5.HLP");   //  set self, help table filename
  help->setTitle(STR_HTITLE);          //Set the Help Window Title
                                    //
  AHelpHandler* phelpHandler=          //Create Custom Help Handler to
    new AHelpHandler();               //  handle the Keys Help
  phelpHandler->handleEventsFor(this);  //Start Help Handler
  return true;                       //
} /* end AHelloWindow :: setupHelp() */

//*************************************************************************
// AHelloWindow :: setupClient()                                           *
//   Setup Client                                                       *
//*************************************************************************
Boolean AHelloWindow :: setupClient()   //Setup Client Window
{                                     //
  clientWindow=new ISplitCanvas(       //Create Canvas
    WND_CANVAS, this, this);          //  with Window Id, parent, owner
  setClient(clientWindow);            //Set canvas as Client Window

  helloCanvas=new ISplitCanvas(        //Create Hello Canvas
    WND_HCANVAS, clientWindow,         //  with Window Id, parent
    clientWindow);                   //  and owner
  helloCanvas->setOrientation(         //Set the orientation
    ISplitCanvas::horizontalSplit);     //  to horizontal

  hello=new IStaticText(WND_HELLO,     //Create Static Text Control
    helloCanvas, helloCanvas);         //  Pass in client as owner & parent
⋮
<Same as Version 4>
⋮
  earthWindow=new AEarthWindow         //Create Earth Graphic Window
    (WND_EARTH, helloCanvas);          //  Set Window ID, client-owner/parent
⋮
<Same as Version 4>
⋮
```

Hello World — Version 5

```
  listBox=new IListBox(WND_LISTBOX,     //Create ListBox
    clientWindow, clientWindow,        //  Parent/Owner is ClientWindow
    IRectangle(),                    //
    IListBox::defaultStyle() |         //
    IControl::tabStop |               //  Set Tab Stop
    IListBox::noAdjustPosition);       //  Allow the Canvas to control size
  listBox->addAscending("Hello, World!");    //Add "Hello, World!"
  listBox->addAscending("Hi, World!");      //Add "Hi, World!"
  listBox->addAscending("Howdy, World!");    //Add "Howdy, World!"
  listBox->addAscending("Alo, Mundo!");     //Add Portuguese Version
  listBox->addAscending("Ola, Mondo!");     //Add Spain
  listBox->addAscending("Hallo wereld!");    //Add Dutch
  listBox->addAscending("Hallo Welt!");     //Add German
  listBox->addAscending("Bonjour le monde!");//Add French
  ISelectHandler::handleEventsFor(listBox);//Set self as select event handler
                                //
  clientWindow->setSplitWindowPercentage(//Set the Window Percentage for
    helloCanvas, 6 );               //  the helloCanvas to 6
  clientWindow->setSplitWindowPercentage(//Set the Window Percentage for
    listBox, 4 );                   //  the listBox to 4
                                //
  return true;                      //
} /* end AHelloWindow :: setupClient() */

//****************************************************************************
// AHelloWindow :: setupInfoArea()                                            *
//   Setup Information Area                                                  *
//****************************************************************************
Boolean AHelloWindow :: setupInfoArea() //Setup Information Area
{                                 //
⋮
<Same as Version 4>
⋮
  setExtensionSize(infoArea,          //
    (int)IFont(infoArea).maxCharHeight());//and specify height
  return true;                      //
} /* end AHelloWindow :: setupInfoArea() */
```

Hello World — Version 5

```
//*************************************************************************
// AHelloWindow :: setupMenuBar()                                         *
//   Setup Menu Bar                                                     *
//*************************************************************************
Boolean AHelloWindow :: setupMenuBar()  //Setup Menu Bar
{                                 //
  ICommandHandler::handleEventsFor(this);//Set self as command event handler
⋮
<Same as Version 4>
⋮
  return true;                    //
} /* end AHelloWindow :: setupMenuBar() */

//*************************************************************************
// AHelloWindow :: setupStatusArea()                                       *
//   Setup Statue Area                                                   *
//*************************************************************************
Boolean AHelloWindow :: setupStatusArea()//Setup Status Area
{                                 //
⋮
<Same as Version 4>
⋮
  return true;                    //
} /* end AHelloWindow :: setupStatusArea() */

//*************************************************************************
// AHelloWindow :: setText(...)                                            *
//   Set Text                                                          *
//*************************************************************************
Boolean AHelloWindow :: setText(const char* text)//Set Text using String
{                                 //
  hello->setText(text);             //Set Text in Control
  return true;                    //Return
} /* end AHelloWindow :: setText(...) */
```

Hello World — Version 5

```
//*************************************************************************
// AHelloWindow :: selected(...)                                          *
//   Handle selected command from list box                                  *
//                                                                       *
// Note: It would be easy to change this selected member function to enter *
//*************************************************************************
Boolean AHelloWindow :: selected(IControlEvent & evt)
{
  IListBox::Cursor lbCursor(*listBox);  //List Box Cursor
  lbCursor.setToFirst();             //Set to first item selected
  setText(listBox->elementAt(lbCursor));//Set Hello Text to Selected Item
  return true;                      //Return Command Processed
} /* end AHelloWindow :: selected(...) */

//*************************************************************************
// AHelloWindow :: command                                                *
//   Handle menu commands                                                *
//*************************************************************************
⋮
<Same as Version 4>
⋮
   case MI_GENERAL_HELP:            //Code to Process Help for help
     help->show(IHelpWindow::general); //Show General Help Panel
     return(true);                 //Return command processed
     break;                       //

⋮
<Same as Version 4>
⋮

//*************************************************************************
// AHelpHandler :: keysHelpId                                             *
//   Handle the keys help request event                                     *
//   This overrides the default provided by IBMCLASS                        *
//*************************************************************************
Boolean AHelpHandler :: keysHelpId(IEvent& evt) //
{                                        //
  evt.setResult(1   );                       //1   =keys help id in
                                         //  ahellow5.ipf file
  return true;                             //Return command processed
} /* end AHelpHandler :: keysHelpId(...) */
```

The AHelloWindow Class Header File

AHELLOW5..HPP contains the class definition and interface specifications for the `AHelloWindow` class, with a few modifications for Version 5. Here is the source listing for AHELLOW5.HPP:

```
#ifndef AHELLOWINDOW_HPP
#define AHELLOWINDOW_HPP

⋮
<Same as Version 4>
⋮
#include <iselhdr.hpp>        //Include ISelectHandler
#include <ihelphdr.hpp>       //Include IHelpHandler

⋮
<Same as Version 4>
⋮
class AEarthWindow;          //Define the AEarthWindow Class

//*************************************************************************
// Class:  AHelloWindow                                                  *
//                                                                      *
// Purpose: Main Window for C++ Hello World sample application          *
//        It is a subclass of IFrameWindow, ICommandHandler and        *
//        ISelectHandler (Processing List Box Selection)              *
//                                                                     *
//*************************************************************************
⋮
<Same as Version 4>
⋮
                public ISelectHandler
⋮
<Same as Version 4>
⋮
   virtual Boolean setupClient();     //Setup Client Window
   virtual Boolean setupHelp();       //Setup Help
   virtual Boolean setupInfoArea() ;  //Setup Information Area
   virtual Boolean setupMenuBar();    //Setup Menu Bar
   virtual Boolean setupStatusArea(); //Setup Status Area

   virtual Boolean setText(const char* text);//Set Text using String

   virtual Boolean selected(IControlEvent& evt);
⋮
<Same as Version 4>
⋮
   IHelpWindow   * help;            //Define Help Window
   AEarthWindow  * earthWindow;      //Define earthWindow
   ISplitCanvas  * clientWindow;     //Define canvas as a Split Canvas
   ISplitCanvas  * helloCanvas;      //Define hello canvas
   IListBox     * listBox;          //Define ListBox
};
```

Hello World — Version 5

```
//*************************************************************************
// Class:  AHelpHandler                                                   *
//                                                                        *
// Purpose: Subclass of IHelpHandler so that the correct keys help       *
//        panel can be displayed when keys help is requested.            *
//                                                                        *
//*************************************************************************
class AHelpHandler: public IHelpHandler//
{                                 //
  protected:                       //Define Protected Member
   virtual Boolean                  //
    keysHelpId(IEvent& evt);         //Override this function
};
#endif
```

The Constants Definition File

AHELLOW5.H contains the constant definitions for this application. These constants and their definitions, which provide the IDs for the application window components, are shown in the following code:

```
#ifndef AHELLOWINDOW_H
#define AHELLOWINDOW_H
//*************************************************************************
// window ids - used by IWindow constructors (eg IStaticText, AHelloWindow)*
//*************************************************************************
⋮
<Same as Version 4>
⋮
#define WND_EARTH        x1 14        //Earth Window ID
#define WND_CANVAS       x1 2         //Canvas Window ID
⋮
<Same as Version 4>
⋮
#define WND_HCANVAS      x1 4         //Hello Canvas Window ID
#define WND_LISTBOX      x1 5         //List Box Window ID


//*************************************************************************
// string ids - used to relate resources to IStaticText and ITitle       *
//*************************************************************************
⋮
<Same as Version 4>
⋮
#define STR_HELPB        x1243        //Help Button String ID
#define STR_HTITLE       x125         //Help Window Title

//*************************************************************************
// menu ids - used on relate command ID to Menu Items and Function Keys   *
//*************************************************************************
⋮
<Same as Version 4>
⋮
#define MI_GENERAL_HELP   x1511        //General Help
```

```
//****************************************************************************
// dialog ids - used to relate dialog fields to controls/commands        *
//****************************************************************************
⋮
<Same as Version 4>
⋮

//****************************************************************************
// help ids - used to relate resources to IHelp Class                  *
//****************************************************************************
#define HELP_TABLE       x18         //Help Table ID
#define SUBTABLE_MAIN     x18 1       //Help Subtable for Main Window
#define SUBTABLE_DIALOG   x18 2       //Help Subtable for Dialog Window
#endif
```

The Text Dialog Source Code File

The ADIALOG5.CPP file contains the source code for the text dialog window constructor, the `ATextDialog` class, created for Version 5.

Here is a listing of the source code:

```
//****************************************************************************
// The entire file was created at version 4                            *
//****************************************************************************

#include <ientryfd.hpp>              //IEntryField Class
#include <icmdevt.hpp>               //ICommandEvent
#include <istring.hpp>               //IString Class
#include <ireslib.hpp>               //IResourceLibrary/IResourceId Class

#include "ahellow5.h"               //Include our Symbolic definitions
#include "adialog5.hpp"              //ATextDialog Class

//****************************************************************************
// ATextDialog :: ATextDialog - Constructor for text dialog window       *
//****************************************************************************
ATextDialog :: ATextDialog(IString & textString, IWindow * ownerWnd)
          : IFrameWindow(IResourceId(WND_TEXTDIALOG), ownerWnd),
            textValue(textString)
{
  ICommandHandler::handleEventsFor(this);//Set self as command event handler

  textValue=textString ;             //Save textValue for exit of dialog
  textField=new IEntryField(DID_ENTRY,  //Create entry field object using dialog
    this);                         //  entry field
  textField->setText(textString);      //Set top current "Hello, World" text
  textField->setFocus();             //Set focus to entry field

} /* end ATextDialog :: ATextDialog(...) */

//****************************************************************************
// ATextDialog :: ~ATextDialog - Destructor                              *
//****************************************************************************
```

Hello World — Version 5

```
ATextDialog :: ~ATextDialog()
{
} /* end ATextDialog :: ~TextDialog(...) */

//*************************************************************************
// ATextDialog :: command - Process Commands                              *
//*************************************************************************
Boolean ATextDialog :: command(ICommandEvent& cmdevt)
{
  switch(cmdevt.commandId()) {
   case DID_OK: // DID_OK           //Process OK Button
     textValue=textField->text() ;    //Get Text from Dialog Entry Field
     dismiss(DID_OK);              //Dismiss Dialog - Allow focus to main
     return(true);                //Return Processing Completed
     break;

   case DID_CANCEL: // DID_CANCEL     //Process CANCEL Button
     dismiss(DID_CANCEL);           //Dismiss Dialog - Allow focus to main
     return(true);                //Return Processing Completed
     break;
  }/* end switch */

  return(false);                 //Allow Default Processing to occur
} /* end ATextDialog :: command(...) */
```

The ATextDialog Class Header File

The ADIALOG5.HPP contains the class definition and interface specifications for the `ATextDialog` class. Here is the source listing for ADIALOG5.HPP:

```
#ifndef ATEXTDIALOG_HPP
#define ATEXTDIALOG_HPP

//*************************************************************************
// The entire file was created at version 4                              *
//*************************************************************************

#include <iframe.hpp>              //IFrameWindow Class (Parent)
#include <icmdhdr.hpp>             //ICommandHandler (Parent)

class IEntryField;

//*************************************************************************
// Class:  ATextDialog                                                   *
//                                                                       *
// Purpose: Dialog window for the C++ Hello World sample application.    *
//        It is a subclass of IFrameWindow, ICommandHandler              *
//                                                                       *
//*************************************************************************
class ATextDialog : public IFrameWindow, public ICommandHandler
{
public:
  ATextDialog (IString & textString, IWindow * ownerWnd) ;
  ~ATextDialog();
```

```
protected:
  virtual Boolean
    command(ICommandEvent& cmdevt);    //Process the dialog command events

private:
  IEntryField * textField ;           //Entry Field to Edit Hello Text
  IString & textValue ;              //String Value for in/out of dialog

}; // TextDialog

#endif
```

The Earth Window Source File

The AEARTHW5.CPP contains the source code for the “earth” window. Here is the source listing for AEARTHW5.CPP:

```
//*************************************************************************
// The entire file was created at version 5                               *
//*************************************************************************

#include <irect.hpp>                //IRectangle Class Header
#include <ipainevt.hpp>              //IPaintEvent Class Header
#include <ihandle.hpp>               //IHandle Class Header

#define INCL_GPIPRIMITIVES           //Set to include GPI Primitives
#define INCL_GPIPATHS               //Set to include GPI Paths
#include <os2.h>

#include "aearthw5.hpp"              //Include our class header

//*************************************************************************
// AEarthWindow :: AEarthWindow - Constructor for "earth" window         *
//*************************************************************************
AEarthWindow :: AEarthWindow(unsigned long windowId,
                        IWindow * parowWindow,
                        const IRectangle& rect) :
             IStaticText(windowId, parowWindow, parowWindow, rect)
{
  handleEventsFor(this);             //Set self as event handler
  show();
} /* end AEarthWindow :: AEarthWindow(...) */

//*************************************************************************
// AEarthWindow :: paintWindow - paint an view of "earth" from space      *
//*************************************************************************
Boolean AEarthWindow :: paintWindow(IPaintEvent & paintEvent)
{
⋮
<Code for painting the window>
⋮
  IPresSpaceHandle hps ;             //Presentation Space Handle

  hps = paintEvent.presSpaceHandle() ;  //Get the presentation space handle
⋮
<More code for painting the window>
```

```
⋮

} /* end AEarthWindow :: paintEvent(..) */
```

The AEarthWindow Class Header File

The AEARTHW5.HPP contains the class definition and interface specifications for the `AEarthWindow` class. Here is the source listing for AEARTHW5.HPP:

```
#ifndef AEARTHWINDOW_HPP
#define AEARTHWINDOW_HPP

//*************************************************************************
// The entire file was created at version 5                               *
//*************************************************************************

#include <istattxt.hpp>             //IStaticText Class Header
#include <ipainhdr.hpp>             //IPaintHandler Class Header

//*************************************************************************
// Class:  AEarthWindow                                                   *
//                                                                        *
// Purpose: "Earth" window for the C++ Hello World sample application.    *
//         It is a subclass of IStaticText & IPaintHandler.               *
//                                                                        *
//*************************************************************************
class AEarthWindow : public IStaticText, public IPaintHandler
{
  public:
    AEarthWindow(unsigned long windowId,//AEarthWindow Constructor
      IWindow * parentownerWindow,     //  Parent/Owner Window
      const IRectangle& rect=IRectangle()); //  Origin/Size Rectangle

    Boolean paintWindow(IPaintEvent&) ; //Handle the paint window event
};
#endif
```

The Resource File

Version 5 of the Hello World application provides a resource file, AHELLOW5.RC. Here is the code used in the AHELLOW5.RC file:

```
⋮
<Same as Version 4>
⋮

//*************************************************************************
// icon and bitmap resources                                             *
//  Symbolic Name (ID) <-> icon filename                                   *
//*************************************************************************
⋮
<Same as Version 4>
```

```
⋮

//*************************************************************************
// string resources - used by IStaticText & ITitle Classes              *
//  Symbolic Name (ID) <-> Text String                                 *
//*************************************************************************
STRINGTABLE
  BEGIN
    STR_HELLO,  "Hello, World!!!!!"          //Hello World String
    WND_MAIN,   "C++ Hello World - Version 5"  //Title Bar String (main id)
⋮
<Same as Version 4>
⋮
    MI_HELP,    "Help Menu"                 //Help Menu ID
    MI_GENERAL_HELP, "Display General Help"    //General Help
    SC_HELPEXTENDED, "Display Extended Help"   //Extended Help
    SC_HELPKEYS,"Display Keys Help"           //Keys Help
    SC_HELPINDEX,"Display Help Index"         //Help Index
⋮
<Same as Version 4>
⋮
    STR_HELPB,  "Help"                     //String for Help Button
    STR_HTITLE, "C++ Hello World - Help Window" //Help Title
  END

//*************************************************************************
// menu bar for main window - used by IMenuBar Class                    *
//  Text String <-> Menu Item ID (Command ID)                          *
//*************************************************************************
⋮
<Same as Version 4>
⋮
    SUBMENU "~Help", MI_HELP                //Help Submenu
      BEGIN
        MENUITEM "~General help...",   MI_GENERAL_HELP
        MENUITEM "~Extented help...",  SC_HELPEXTENDED, MIS_SYSCOMMAND
        MENUITEM "~Keys help...",     SC_HELPKEYS, MIS_SYSCOMMAND
        MENUITEM "Help ~index...",    SC_HELPINDEX, MIS_SYSCOMMAND
      END
  END

//*************************************************************************
// Accelerator (key) table resources - used by IAccelerator Class       *
//  Key Value <-> Menu Item ID (Command ID)                           *
//*************************************************************************
⋮
<Same as Version 4>
⋮

//*************************************************************************
// dialog resources - used by ATextDialog Class                         *
//*************************************************************************
⋮
<Same as Version 4>
⋮
```

Hello World — Version 5

```
//*************************************************************************
// help table resources - used by IHelp Class                             *
//*************************************************************************
HELPTABLE HELP_TABLE
  BEGIN
    HELPITEM WND_MAIN,       SUBTABLE_MAIN,   1
    HELPITEM WND_TEXTDIALOG,  SUBTABLE_DIALOG, 2
  END

HELPSUBTABLE SUBTABLE_MAIN                  //Main Window Help Subtable
  BEGIN                                  //
    HELPSUBITEM WND_HELLO, 1                 //Hello <-> Help ID 1
    HELPSUBITEM WND_LISTBOX,1 2              //List Box Help
    HELPSUBITEM MI_EDIT, 11                 //Edit Menu
    HELPSUBITEM MI_ALIGNMENT, 111             //Alignment Menu
    HELPSUBITEM MI_LEFT, 112                //Left Menu Item
    HELPSUBITEM MI_CENTER, 113               //Center Menu Item
    HELPSUBITEM MI_RIGHT, 114                //Right Menu Item
    HELPSUBITEM MI_TEXT, 199                 //Text Menu Item
  END                                    //

HELPSUBTABLE SUBTABLE_DIALOG                 //Text Dialog Help Subtable
  BEGIN                                  //
    HELPSUBITEM DID_ENTRY, 2 1               //Entry Field <-> Help ID 2 1
    HELPSUBITEM DID_OK, 2 2                 //OK Button <-> Help ID 2 2
    HELPSUBITEM DID_CANCEL, 2 3              //OK Button <-> Help ID 2 3
  END                                    //
```

The Icon File

AHELLOW5.ICO is used as both the title bar icon and the icon that is displayed when the application is minimized. We cannot provide a listing for the AHELLOW5.ICO file, but this is how the icon appears when minimized:

Hello World Icon

Figure 55. Hello World Icon

The Text Dialog Template

ADIALOG5.DLG contains the template used to build the text dialog. Here is that template:

```
DLGINCLUDE 1 "AHELLOW5.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, 4 , 14
      PUSHBUTTON     "Cancel", DID_CANCEL, 49, 4, 4 , 14
      LTEXT         "Edit Text:", DID_STATIC, 8, 62, 69, 8
      ENTRYFIELD     "", DID_ENTRY, 8, 44, 114, 8, ES_MARGIN
   END
END
```

The Text Dialog Resource File

ADIALOG5.RES is created by the resource compiler as input to HELLO5.EXE.

The Help Window Source File

The AHELLOW5.IPF file contains the text and IPF tags used to produce the help information for the Hello World application. IPF, or Information Presentation Facility, uses a tag language to format the text that appears in a help window. For example, `:p.` is the paragraph tag, which is used to start a new paragraph. Refer to the *OS/2 2.0 Information Presentation Facility Guide and Reference* for descriptions of other tags used in the following source file. The IPFC complier, provided by the OS/2 2.0 Developer's Toolkit, is used to compile this file.

```
:userdoc.
:docprof toc=123456.
:title.C++ Hello World Help
:h1.C++ Hello World - Application Help
:p.This file contains the help for the C++ Hello World Application.
:h2 res=1  .C++ Hello World - Main Window Help
:p.This is the help panel for the main window.
The main window contains the following areas:
:ul.
:li.The title bar icon, which provides access to the system menu.
:li.The window title, which displays the title of the window.
:li.The menu bar, which allows the user to select specific actions.
:li.A status line, which contains the current alignment.
:li.A client area, which is divided into three areas:
:ul.
:li.The first area contains the static text for :q.Hello, World:eq..
```

Hello World — Version 5

```
:li.The second area is a graphic control that shows a graphic of the world
from space with stars.
:li.The third area contains a list box that allows the user to change the
:q.Hello, World:eq. text string.
:eul.
:li.Alignment push buttons, which change the alignment, and a help push
button, which is used to request help.
:li.An information area, which helps the user understand the current options
of the program, including the menu bar choices.
:eul.
:h2 res=1 2.C++ Hello World - List Box Help
:p.This is the help panel for the list box window.
Selecting any item in the list box changes the :q.Hello, World:eq. text.
The code that handles the list box can be found in AHELLOW5.CPP.
```

Hello World — Version 5

```
:h2 res=11 .C++ Hello World - Edit Menu Help
:p.This is the help panel for the Edit menu.
:p.
This submenu (MI_EDIT) can be found under the following statement in the
resource file (AHELLOW5.RC):
:xmp.
   SUBMENU "~Edit", MI_EDIT             //Edit Submenu
:exmp.
:p.
This help panel (id=11 ) was linked to the menu item (MI_EDIT)
by the following statement in the resource file (AHELLOW5.RC):
:xmp.
   HELPSUBITEM MI_EDIT, 11           //Edit Menu
:exmp.
:h3 res=111.C++ Hello World - Alignment Menu Help
:p.This is the help panel for the Alignment menu item.
:p.
This submenu (MI_ALIGNMENT) can be found under the following statement in the
resource file (AHELLOW5.RC):
:xmp.
   SUBMENU "~Alignment", MI_ALIGNMENT     //Alignment Submenu
:exmp.
:p.
This help panel (id=111) was linked to the menu item (MI_ALIGNMENT)
by the following statement in the resource file (AHELLOW5.RC):
:xmp.
   HELPSUBITEM MI_ALIGNMENT, 111      //Alignment Menu
:exmp.
:h4 res=112.C++ Hello World - Left Command Help
:p.This is the help panel for the Left alignment command.
Selecting the Left menu item or Left push button sets the :q.Hello, World:eq.
text to be left aligned.
:p.
This menu item (MI_LEFT) was created by the following statement in the
resource file (AHELLOW5.RC):
:xmp.
   MENUITEM "~Left\tF7",   MI_LEFT    //Left Menu Item - F7 Key
:exmp.
:p.
The code that handles this menu item can be found in AHELLOW5.CPP under the
following case statement:
:xmp.
   case MI_LEFT:                   //Code to Process Left Command Item
:exmp.
:p.
This help panel (id=112) was linked to the menu item (MI_LEFT)
by the following statement in the resource file (AHELLOW5.RC):
:xmp.
   HELPSUBITEM MI_LEFT, 112          //Left Menu Item
:exmp.
```

Hello World — Version 5

```
:h4 res=113.C++ Hello World - Center Command Help
:p.This is the help panel for the Center alignment command.
Selecting the Center menu item or Center push button sets the
:q.Hello, World:eq. text to be center aligned.
:p.
This menu item (MI_CENTER) was created by the following statement in the
resource file (AHELLOW5.RC):
:xmp.
   MENUITEM "~Center\tF8", MI_CENTER  //Center Menu Item - F8 Key
:exmp.
:p.
The code that handles this menu item can be found in AHELLOW5.CPP under the
following case statement:
:xmp.
   case MI_CENTER:                //Code to Process Center Command Item
:exmp.
:p.
This help panel (id=113) was linked to the menu item (MI_CENTER)
by the following statement in the resource file (AHELLOW5.RC):
:xmp.
   HELPSUBITEM MI_CENTER, 113        //Center Menu Item
:exmp.
:h4 res=114.C++ Hello World - Right Command Help
:p.This is the help panel for the Right alignment command.
Selecting the Right menu item or Right push button sets the
:q.Hello, World:eq. text to be right aligned.
:p.
This menu item (MI_RIGHT) was created by the following statement in the
resource file (AHELLOW5.RC):
:xmp.
   MENUITEM "~Right\tF9",  MI_RIGHT   //Right Menu Item - F9 Key
:exmp.
:p.
The code that handles this menu item can be found in AHELLOW5.CPP under the
following case statement:
:xmp.
   case MI_RIGHT:                 //Code to Process Right Command Item
:exmp.
:p.
This help panel (id=114) was linked to the menu item (MI_RIGHT)
by the following statement in the resource file (AHELLOW5.RC):
:xmp.
   HELPSUBITEM MI_RIGHT, 114         //Right Menu Item
:exmp.
```

Hello World — Version 5

```
:h3 res=199.C++ Hello World - Text... Menu Help
:p.This is the help panel for the Text... menu item.
:p.
This menu item (MI_TEXT) was created by the following statement in the
resource file (AHELLOW5.RC):
:xmp.
   MENUITEM "~Text...", MI_TEXT          //Text Menu Item
:exmp.
:p.
The code that handles this menu item can be found in AHELLOW5.CPP under the
following case statement:
:xmp.
   case MI_TEXT:                   //Code to Process Text Command
:exmp.
:p.
This help panel (id=199) was linked to the menu item (MI_TEXT)
by the following statement in the resource file (AHELLOW5.RC):
:xmp.
   HELPSUBITEM MI_TEXT, 199          //Right Menu Item           .
:exmp.
:h2 res=2  .C++ Hello World - Dialog Window Help
:p.This is the help panel for the text dialog.
:h3 res=2 1.C++ Hello World - Dialog Entry Field Help
:p.This is the help panel for the entry field in the text dialog.
:p.
This entry field was defined by the following statement in the dialog
resource file (ADIALOG5.DLG):
:xmp.
   ENTRYFIELD     "", DID_ENTRY, 8, 44, 114, 8, ES_MARGIN
:exmp.
:p.
The application code that handles this entry field can be found in
ADIALOG5.CPP.
:p.
This help panel (id=2 1) was linked to this entry field (DID_ENTRY)
by the following statement in the resource file (AHELLOW5.RC):
:xmp.
   HELPSUBITEM DID_ENTRY, 2 1        //Entry Field <-> Help ID 2 1
:exmp.
```

Hello World — Version 5

```
:h3 res=2 2.C++ Hello World - Dialog OK Button Help
:p.This is the help panel for the OK push button in the text dialog.
Selecting this push button closes the dialog.
Any changes made to the :q.Hello, World:eq. text are shown in the main
window.
:p.
The OK push button is defined by the following statement in the dialog
resource file (ADIALOG5.DLG):
:xmp.
   DEFPUSHBUTTON   "OK", DID_OK, 6, 4, 4 , 14
:exmp.
:p.
The application code that handles this push button field can be found
in ADIALOG5.CPP.
:p.
This help panel (id=2 2) was linked to this entry field (DID_OK)
by the following statement in the resource file (AHELLOW5.RC):
:xmp.
   HELPSUBITEM DID_OK, 2 2          //OK Button <-> Help ID 2 2
:exmp.
:h3 res=2 3.C++ Hello World - Dialog Cancel Button Help
:p.This is the text for the Cancel push button in the text dialog.
Selecting this button closes the dialog without changing the
:q.Hello, World:eq. text.
:p.
The Cancel push button is defined by the following statement in the dialog
resource file (ADIALOG5.DLG):
:xmp.
   PUSHBUTTON     "Cancel", DID_CANCEL, 49, 4, 4 , 14
:exmp.
:p.
The application code that handles this push button field can be found
in ADIALOG5.CPP.
:p.
This help panel (id=2 3) was linked to this entry field (DID_CANCEL)
by the following statement in the resource file (AHELLOW5.RC):
:xmp.
   HELPSUBITEM DID_CANCEL, 2 3       //OK Button <-> Help ID 2 3
:exmp.
```

```
:h2 res=1   .C++ Hello World - Keys Help Panel
:p.This is the keys help panel.
:p.The following is a list of system-provided keys:
:dl compact tsize=1 .
:dt.Alt-F4
:dd.Close window.
:dt.Alt-F7
:dd.Move window.
:dt.Alt-F8
:dd.Size window.
:dt.Alt-F9
:dd.Minimize window.
:dt.Alt-F1
:dd.Maximize window.
:edl.
:p.The following is a list of application-provided keys:
:dl compact tsize=1 .
:dt.F7
:dd.Left alignment.
:dt.F8
:dd.Center alignment.
:dt.F9
:dd.Right alignment.
:edl.
:euserdoc.
```

The Module Definition File

The AHELLOW5.DEF file is required for the same reasons that AHELLOW1.DEF was needed for Version 1. See “The Module Definition File” on page 148 if you need to review the reasons for creating a .DEF file.

The only differences between the two .DEF files used in Version 1 and Version 5 are the change in the version number and the stack size.

```
NAME    HELLO5    WINDOWAPI

DESCRIPTION 'Hello World Sample C++ Program - Version 5'

CODE    LOADONCALL MOVEABLE
DATA    MOVEABLE   MULTIPLE

HEAPSIZE  8192
STACKSIZE 128
```

Tasks Performed by Version 5

The following sections describe each of the tasks performed by Version 5 of the Hello World application that were not described already for previous versions. Those tasks use newly defined member functions to construct the main window. The tasks performed by those member functions are:

- Setting up the client window with the following:
  - Two split canvases, one split vertically in the client window and one split horizontally in the left pane of the first split canvas
  - The static text control used to display the “Hello, World!!!!!” text string, placed in the top pane of the horizontally split canvas
  - A new control (`AEarthWindow`) that uses PM graphics calls to paint one of the panes in the split canvas, placed in the bottom pane of the horizontally split canvas
  - A list box containing text strings that can be used to replace the “Hello, Window!!!!!” text string, placed in the right pane of the vertically split canvas
- Adding help windows for the main, dialog, and entry field windows
- Adding a help handler (`AHelpHandler`) to display the correct window when keys help is requested
- Setting up the information area, menu bar, and status area

Tasks this version performs that were described for Version 4 are:

- Adding a cascaded menu to a pull-down menu
- Adding accelerator, or shortcut, keys for the Left (F7), Center(F8), and Right(F9) cascaded menu choices
- Adding the `ADialogText` class to allow the user to edit the “Hello, World!!!!” text string
- Adding push buttons in a set canvas

Tasks this version performs that were described for Version 3 are:

- Creating a status area using a static text control
- Putting text in a static text control for a status line
- Specifying the location and height of the status area
- Setting `AHelloWindow` as the event handler

- Creating a menu bar
- Setting an initial check mark in the pull-down menu
- Adding command processing to set the static text control alignment

Tasks this version performs that were described for Versions 1 and 2 are:

- Creating the main window
- Getting the current application and running it
- Constructing the main window, which involves the following:
  - Creating a static text control
  - Setting a text string from a resource file
  - Putting a text string into a static text control
  - Aligning the text
  - Setting the static text control in the main window
  - Setting the window title and title bar icon from a resource file
  - Creating and setting the information area below the client area
  - Setting the focus to the main window and showing the main window

Constructing the Main Window Using Newly Defined Member Functions

Version 5 provides several new member functions for the `AHelloWindow` class to use when constructing the main window. They are declared as protected member functions in the AHELLOW5.HPP file:

```
//<in AHELLOW5.HPP>
protected:                          //Define Protected Member
  virtual Boolean setupClient();     //Setup Client Window
  virtual Boolean setupHelp();       //Setup Help
  virtual Boolean setupInfoArea();   //Setup Information Area
  virtual Boolean setupMenuBar();     //Setup Menu Bar
  virtual Boolean setupStatusArea();  //Setup Status Area
```

These member functions are implemented in the `AHelloWindow` window constructor in AHELLOW5.CPP, as follows:

```
//<in AHELLOW5.CPP>
setupClient();                    //Setup Client Window              v5
setupStatusArea();                 //Setup Status Area                .
setupInfoArea();                   //Setup Information Area           v5
setupMenuBar();                    //Setup Menu Bar                  v5
setupHelp();                      //Setup Help                      v5
```

The following sections describe the implementation of these functions.

Setting Up the Client Window

The `setupClient` member function sets up the client window for the main window, as follows:

- Creates a split canvas control as the client window with the canvas split vertically into a left pane and a right pane by default:

```
//<in AHELLOW5.CPP>
clientWindow=new ISplitCanvas(       //Create Canvas                  .
  WND_CANVAS, this, this);           //  with Window Id, parent, owner     .
setClient(clientWindow);            //Set canvas as Client Window        .
```

- Creates another split canvas control in the left pane of the first split canvas control; the second canvas is split horizontally:

```
//<in AHELLOW5.CPP>
helloCanvas=new ISplitCanvas(        //Create Hello Canvas              .
  WND_HCANVAS, clientWindow,         //  with Window Id, parent         .
  clientWindow);                   //  and owner                   .
helloCanvas->setOrientation(         //Set the orientation            .
  ISplitCanvas::horizontalSplit);    //  to horizontal                v5
```

- Puts the static text control for displaying the “Hello, World!!!!!” text string in the top pane of the second split canvas, sets the text in it, and centers the text:

```
//<in AHELLOW5.CPP>
hello=new IStaticText(WND_HELLO,     //Create Static Text Control
  helloCanvas, helloCanvas);       //  Pass in client as owner & parent  v5
hello->setText(STR_HELLO);           //Set text                        v2
hello->setAlignment(               //Set Alignment to Center in both
  IStaticText::centerCenter);        //   directions
```

- Creates a graphic in the bottom pane of the second split canvas using the `AEarthWindow` class, which is new for Version 5:

```
//<in AHELLOW5.CPP>
earthWindow=new AEarthWindow         //Create Earth Graphic Window       v5
  (WND_EARTH, helloCanvas);        //  Set Window ID, client-owner/parentv5
```

  The interface specifications and implementation of the `AEarthWindow` class are declared in the AEARTHW5.HPP file, shown in “The AEarthWindow Class Header File” on page 230, and in the AEARTHW5.CPP file, shown in “The Earth Window Source File” on page 229.

- Creates a list box in the right pane of the client window split canvas and fills the list box with text strings:

```
//<in AHELLOW5.CPP>
listBox=new IListBox(WND_LISTBOX,    //Create ListBox                  v5
```

```
  clientWindow, clientWindow,        //  Parent/Owner is ClientWindow      .
  IRectangle(),                    //                                .
  IListBox::defaultStyle() |         //                                  .
  IControl::tabStop |               //  Set Tab Stop                     .
  IListBox::noAdjustPosition);       //  Allow the Canvas to control size   .
listBox->addAscending("Hello, World!");    //Add "Hello, World!"           .
listBox->addAscending("Hi, World!");      //Add "Hi, World!"              .
listBox->addAscending("Howdy, World!");    //Add "Howdy, World!"           .
listBox->addAscending("Alo, Mundo!");      //Add Portuguese Version        .
listBox->addAscending("Ola, Mondo!");      //Add Spain                    .
listBox->addAscending("Hallo wereld!");    //Add Dutch                    .
listBox->addAscending("Hallo Welt!");      //Add German                   .
listBox->addAscending("Bonjour le monde!");//Add French                    .
ISelectHandler::handleEventsFor(listBox);//Set self as select event handler  .
```

- Allocates 60 percent of the screen for the left pane of the client window split canvas, and 40 percent for the right pane:

```
//<in AHELLOW5.CPP>
clientWindow->setSplitWindowPercentage(//Set the Window Percentage for      .
  helloCanvas, 6 );                //  the helloCanvas to 6             .
clientWindow->setSplitWindowPercentage(//Set the Window Percentage for      .
  listBox, 4 );                    //  the listBox to 4                 .
```

Setting Up Help

The `setupHelp` member function sets up the help area, as follows:

- Creates a help window:

```
//<in AHELLOW5.CPP>
help=new IHelpWindow(HELP_TABLE,      //Create Help Window Object          .
  this);                          //Setup Help info                   .
```

  The HELP_TABLE constant identifies the following help table in the resource file, AHELLOW5.RC:

```
//<in AHELLOW5.RC>
HELPTABLE HELP_TABLE                                                  // .
  BEGIN                                                             // .
    HELPITEM WND_MAIN,       SUBTABLE_MAIN,   1                          // .
    HELPITEM WND_TEXTDIALOG, SUBTABLE_DIALOG, 2                            // .
  END                                                              //v5
```

  This help table provides help for the main window (WND_MAIN) and also for the text dialog (WND_TEXTDIALOG) that is used to edit the "Hello, World!!!!!" text string (see Chapter 14, "Simple Dialogs and Push Buttons" on page 189 for a description of the text dialog). The window IDs for WND_MAIN and WND_TEXTDIALOG are specified in AHELLOW5.H:

```
//<in AHELLOW5.H>
#define WND_MAIN        x1          //Main Window ID
#define WND_TEXTDIALOG   x1 13       //Text Dialog Window ID           v4
```

Hello World — Version 5

The SUBTABLE_MAIN and SUBTABLE_DIALOG constants identify two help subtables, which define other windows, plus menu items, the entry field in the text dialog, and push buttons for which help is available:

```
//<in AHELLOW5.RC>
HELPSUBTABLE SUBTABLE_MAIN                  //Main Window Help Subtable   v5
  BEGIN                                  //                          .
    HELPSUBITEM WND_HELLO, 1                 //Hello <-> Help ID 1        .
    HELPSUBITEM WND_LISTBOX,1 2              //List Box Help
    HELPSUBITEM MI_EDIT, 11                 //Edit Menu                .
    HELPSUBITEM MI_ALIGNMENT, 111            //Alignment Menu            .
    HELPSUBITEM MI_LEFT, 112                //Left Menu Item            .
    HELPSUBITEM MI_CENTER, 113               //Center Menu Item          .
    HELPSUBITEM MI_RIGHT, 114                //Right Menu Item           .
    HELPSUBITEM MI_TEXT, 199                 //Text Menu Item            .
  END                                    //                       v5

HELPSUBTABLE SUBTABLE_DIALOG                  //Text Dialog Help Subtable   v5
  BEGIN                                  //                          .
    HELPSUBITEM DID_ENTRY, 2 1               //Entry Field <-> Help ID 2 1  .
    HELPSUBITEM DID_OK, 2 2                 //OK Button <-> Help ID 2 2   .
    HELPSUBITEM DID_CANCEL, 2 3              //OK Button <-> Help ID 2 3   .
  END                                    //                       v5
```

- Designates the AHELLOW5.HLP file as the source of the help information:

```
//<in AHELLOW5.CPP>
help->addLibraries("AHELLOW5.HLP");  //  set self, help table filename     .
```

The IPFC compiler, which is included with the OS/2 2.0 Developer's Toolkit, is used to compile the AHELLOW5.IPF file to produce the AHELLOW5.HLP file. See "The Help Window Source File" on page 233 for the AHELLOW5.IPF source listing. The help provided for the main window of the Hello World application looks like this:

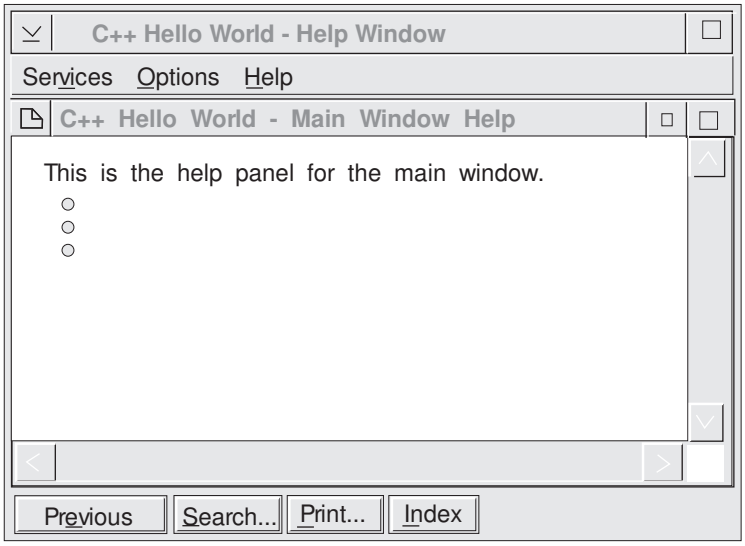


Figure 56. Main Window Help for Hello World Version 5

This window is displayed by pressing the F1 key, or selecting the Help choice on the menu bar and then selecting General help... from the pull-down menu, The following code is used:

```
//<in AHELLOW5.CPP>
case MI_GENERAL_HELP:            //Code to Process Help for help      v5
  help->show(IHelpWindow::general); //Show General Help Panel           .
  return(true);                 //Return command processed          .
  break;                       //                                v5
```

- Sets the title of the help window:

```
//<in AHELLOW5.CPP>
help->setTitle(STR_HTITLE);         //Set the Help Window Title         .
```

- Creates a handler, `AHelpHandler` to customize the keys help:

```
//<in AHELLOW5.CPP>
AHelpHandler* phelpHandler=         //Create Custom Help Handler to      .
  new AHelpHandler();              //  handle the Keys Help            .
```

The interface specifications for the `AHelpHandler` class are declared in the AHELLOW5.HPP header file:

Hello World — Version 5

```
//<in AHELLOW5.HPP>
class AHelpHandler: public IHelpHandler//                                .
{                                //                              .
  protected:                      //Define Protected Member           .
   virtual Boolean                 //                              .
    keysHelpId(IEvent& evt);         //Override this function            .
};                                                            //v5
```

The `keysHelpId` member function of `AHelpHandler` is implemented in the AHELLOW5.CPP file as follows:

```
//<in AHELLOW5.CPP>
Boolean AHelpHandler :: keysHelpId(IEvent& evt) //                        .
{                                        //                      .
  evt.setResult(1   );                       //1   =keys help id in      .
                                         //  AHELLOW5.IPF file       .
  return true;                             //Return command processed   .
} /* end AHelpHandler :: keysHelpId(...) */                           //v5
```

- Starts the help handler:

```
//<in AHELLOW5.CPP>
phelpHandler->handleEventsFor(this);  //Start Help Handler               .
```

Setting Up the Information Area

The `setupInfoArea` member function sets up the information area for the main window, as follows:

- Creates the information area:

```
//<in AHELLOW5.CPP>
infoArea=new IInfoArea(this);       //Create the information area       v2
```

- Puts the text in the information area from the resource file as in previous versions:

```
//<in AHELLOW5.CPP>
infoArea->setInactiveText(STR_INFO);  //Set information area text from RC   v2
```

- Uses the height of the current font as the height of the information area:

```
//<in AHELLOW5.CPP>
setExtensionSize(infoArea,          //                              v5
  (int)IFont(infoArea).maxCharHeight());//and specify height              .
```

Setting Up the Menu Bar

The `setupMenuBar` member function sets up the menu bar for the main window, as follows:

- Sets the main window as the event handler for commands:

```
// <in AHELLOW5.CPP>
ICommandHandler::handleEventsFor(this);//Set self as command event handler  v5
```

- Creates the menu bar:

```
// <in AHELLOW5.CPP>
menuBar=new IMenuBar(WND_MAIN,       //Create Menu Bar for main window    v3
  this);                         //  Set self as parent              .
```

- Places a check on the Center choice in the cascading menu that is displayed when the Alignment choice is selected from the Edit menu:

```
// <in AHELLOW5.CPP>
menuBar->checkItem(MI_CENTER);       //Place Check on Center Menu Item    v3
```

Setting Up the Status Area

The `setupStatusArea` member function sets up the status area for the main window, as follows:

- Creates the status area:

```
// <in AHELLOW5.CPP>
statusLine=new IStaticText          //Create Status Area using Static Textv3
  (WND_STATUS, this, this);         //  Window ID, Parent, Owner Parameters.
```

- Gets the “Center Alignment” text string from the resource file and sets it in the status area:

```
// <in AHELLOW5.CPP>
statusLine->setText(STR_CENTER);     //Set Status Text to "Center" from Res .
```

- Sets the position and height of the status area. The status area is placed above the client area and its height is that of the current font:

```
// <in AHELLOW5.CPP>
addExtension(statusLine,           //Add Status Line                 .
  IFrameWindow::aboveClient,        //  above the client area           .
  IFont(statusLine).maxCharHeight()); //  and specify height             v3
```

Compiling and Linking Version 5

Figure 57 shows the files that were used to create Version 5 of the Hello World application, their relationship to each other, and the order in which they are compiled and linked. File names are shown in uppercase letters; program names are shown in lowercase letters and are enclosed in a rectangle.

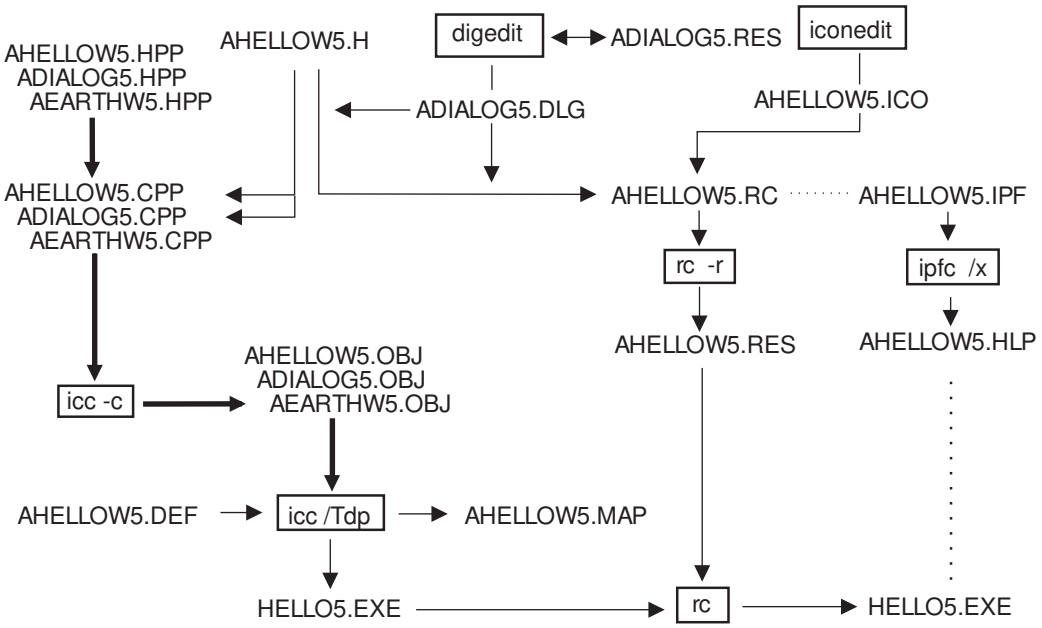


Figure 57. Compiling and Linking Version 5 of the Hello World Application

Hello World — Version 5

Chapter 16. NLS and Advanced Functions

Version 6 of the Hello World application covers the following topics:

- Specifying English, German, or Portuguese DLL resources using the command line
- Adding an **Open...** menu item and using a file dialog
- Showing a message box when the input file cannot be read from the file dialog
- Adding a pop-up menu for changing the alignment

© Copyright IBM Corp. 1993

Hello World — Version 6

- Changing the status area to a split canvas and adding the date and time
- Adding a time handler (`ATimeHandler`) and updating the time on the status area
- Adding HELLOWPS.CMD to create a folder with programs on the OS/2 Workplace Shell

The window for Version 6 of the Hello World application looks like this:

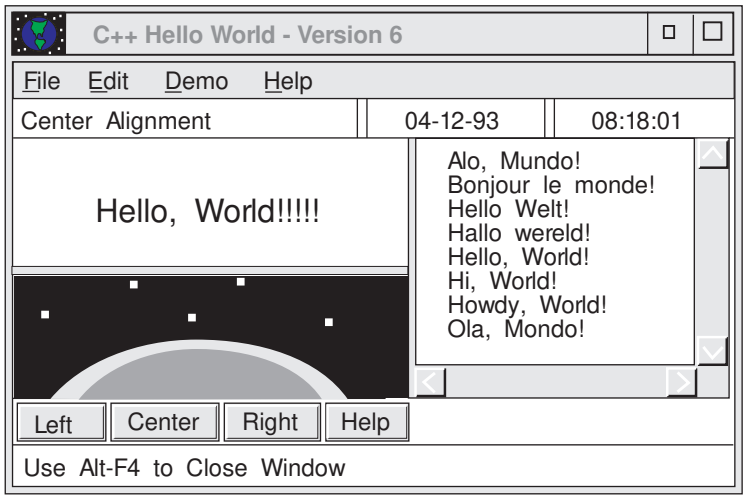


Figure 58. Version 6 of Hello World Application

Version 6 Window-Parent Relationship Diagram

Figure 59 shows the relationship between the objects built for Version 6 of the Hello World application:

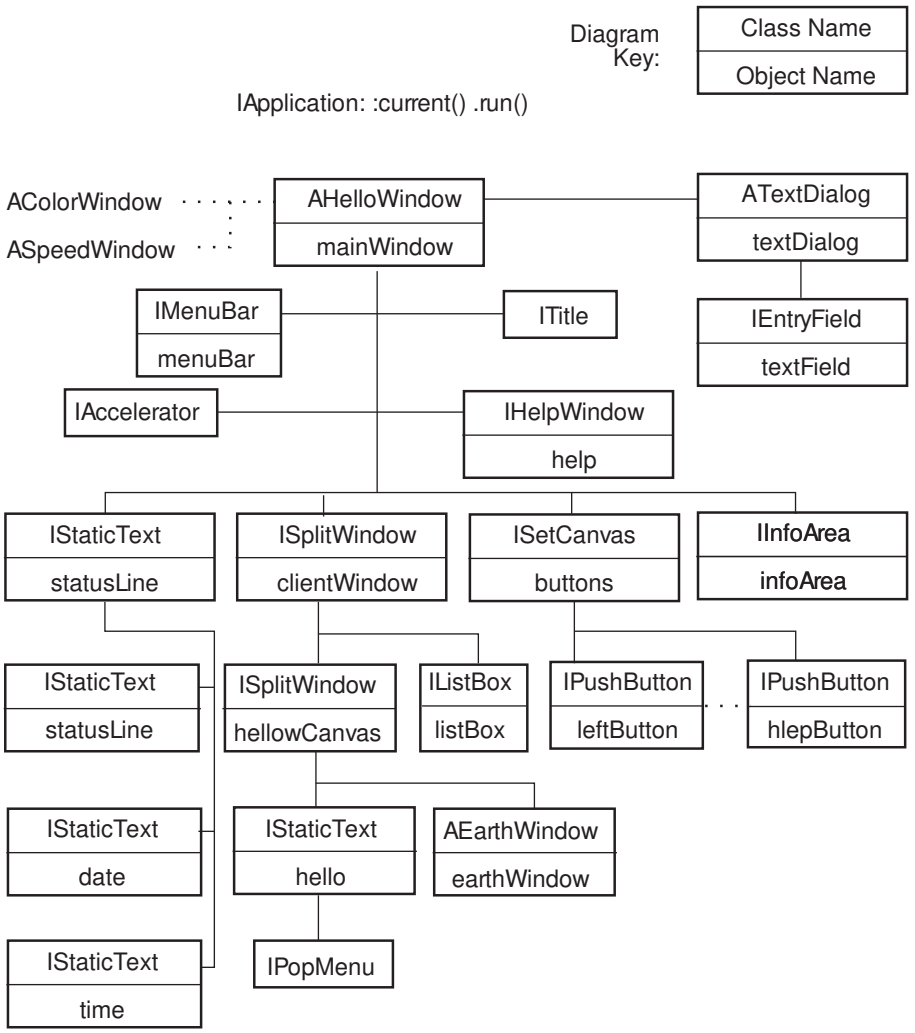


Figure 59. Window Parent Relationship Diagram

Version 6 Files

The following files contain the code used to create this version:

AHELLOW6.CPP	Source code for main procedure and `AHelloWindow` class.
AHELLOW6.HPP	Header file for the `AHellowWindow` class.
AHELLOW6.H	Symbolic definition file for HELLO6.EXE.
ADIALOG6.CPP	Source code to create the `ATextDialog` class.
ADIALOG6.HPP	Header file for the `ATextDialog` class.
AEARTHW6.CPP	Source code to create the `AEarthWindow` class.
AEARTHW6.HPP	Header file for the `AEarthWindow` class.
ACOLORW6.CPP	Source code to create the `AColorWindow` class.
ACOLORW6.HPP	Header file for the `AColorWindow` class.
ASPEEDW6.CPP	Source code to create the `ASpeedWindow` class.
ASPEEDW6.HPP	Header file for the `ASpeedWindow` class.
ATIMEHDR.CPP	Source code to create the `ATimeHandler` class.
ATIMEHDR.HPP	Header file for the `ATimeHandler` class.
ADUMMY6.CPP	File to provide dummy file for resource DLLs.
AHELLOWE.RC	English resource file for HELLO5.EXE.
AHELLOWG.RC	German resource file for HELLO6.EXE.
AHELLOWP.RC	Portuguese resource file for HELLO6.EXE.
AHELLOW6.ICO	Icon file for HELLO6.EXE.
BRAZIL.ICO	Icon file for Portuguese option of HELLO6.EXE.
GERMANY.ICO	Icon file for German option of HELLO6.EXE.
ADIALOGE.DLG	English dialog resource source file for HELLO6.EXE.
ADIALOGG.DLG	German dialog resource source file for HELLO6.EXE.

ADIALOGP.DLG	Portuguese dialog resource source file for HELLO6.EXE.
ADIALOGE.RES	Dialog resource file for HELLO6.EXE.
AHELLOW6.IPF	Help file for HELLO6.EXE.
AHELLOW6.DEF	Module definition file for HELLO6.EXE.
AHELLOWE.DEF	Module definition file for AHELLOWE.DLL.
AHELLOWG.DEF	Module definition file for AHELLOWG.DLL.
AHELLOWP.DEF	Module definition file for AHELLOWP.DLL.

Tasks Performed by Version 6

The listings of the source files are not included in this book due to the size. The following list contains the changes required for each key task:

- Using English, German or Portuguese DLL resources
  - Updated main routine in AHELLOW6.CPP
  - Created AHELLOWE.RC, AHELLOWG.RC, and AHELLOWP.RC resource files.
  - Created ADIALOGE.DLG, ADIALOGG.DLG, and ADIALOGP.DLG dialog files.
  - Created BRAZIL.ICO and GERMAN.ICO icon files.
- Adding an **Open...** menu item and using a file dialog
  - Updated Menu in resource files
  - Added `openFile` member function to AHELLOW6.CPP and AHELLOW6.HPP
- Showing a message box
  - Added code in the `openFile` member function
  - Added STR_MSGTXT string resource to resource files
- Adding a pop-up menu for changing the alignment
  - Added a new menu to the resource files

  - Added the `AMenuHandler` class with the `makePopUpMenu` member function in AHELLOW6.CPP and AHELLOW6.HPP
  - Updated `setupClient` in AHELLOW6.CPP to create this handler and attach it to the hello static text window.
- Changing the status area to a split canvas and adding the date and time
  - Updated the `setupStatusArea` member function in AHELLOW6.CPP
- Adding a time handler and updating the time on the status area
  - Added the `ATimeHandler` class in the new ATIMEHDR.CPP and ATIMEHDR.HPP files
  - Added `ATimeHandler::handleEventsFor(this);` in the constructor for `AHelloWindow`
  - Added `ATimeHandler::stopHandlingEventsFor(this);` in the destructor for `AHelloWindow`
  - Added the `tick` member function in AHELLOW6.CPP and AHELLOW6.HPP

Compiling and Linking Version 6

Figure 60 on page 257 shows the files used to create Version 6 of the Hello World application, their relationship to each other, and the order in which they are compiled and linked. File names are shown in uppercase letters; program names are shown in lowercase letters and are enclosed in a rectangle.

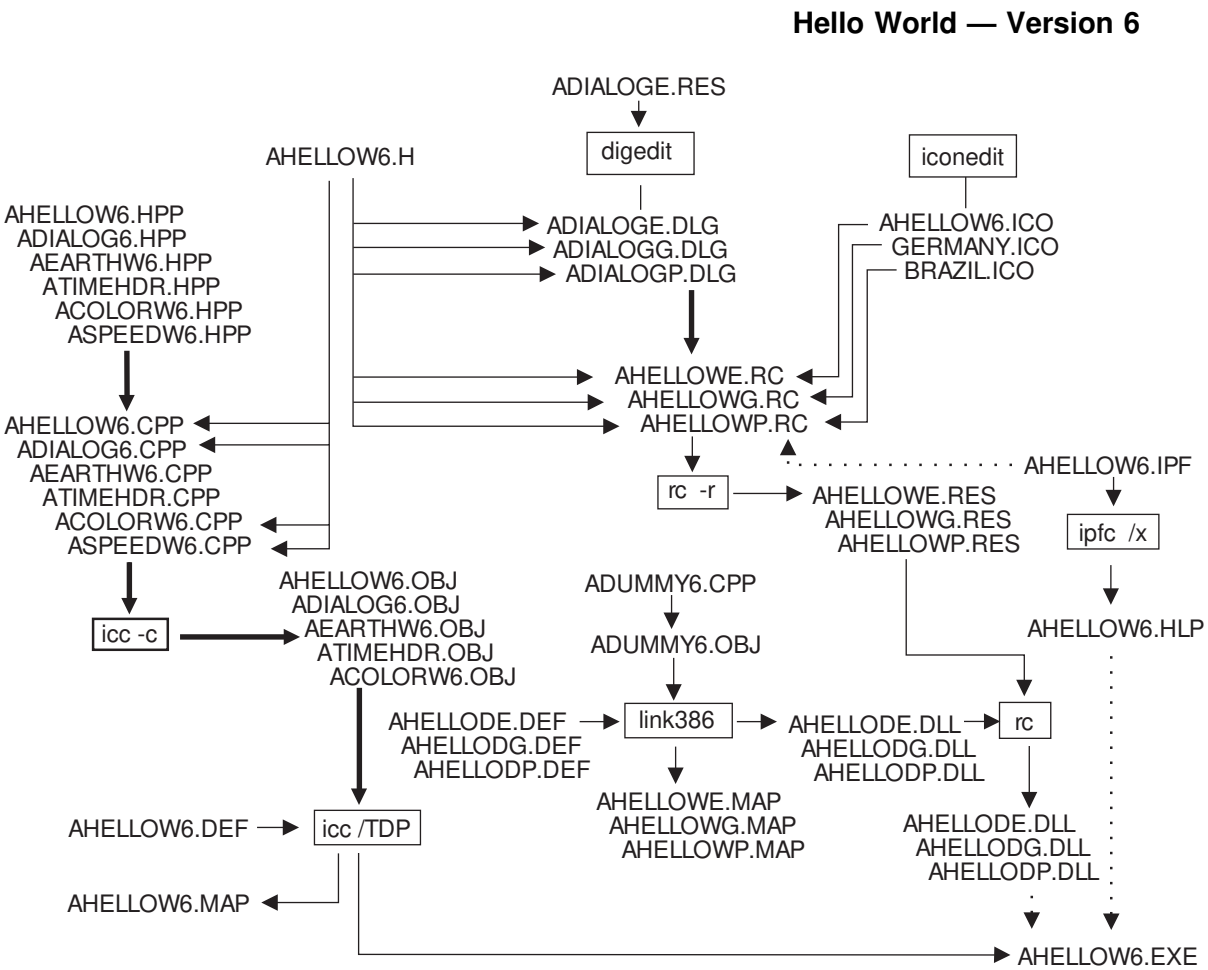


Figure 60. Compiling and Linking Version 6 of the Hello World Application

Hello World — Version 6

Appendix A. Hierarchy

Application Classes

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

© Copyright IBM Corp. 1993

Window Classes

```
IBase
   IMessageBox
   IVBase
      IWindow
         IControl
            ICanvas
               IMultiCellCanvas
               ISetCanvas
               ISplitCanvas
               IViewPort
            IContainerControl
            IListBox
            INotebook
            IOutlineBox
            IProgressIndicator
               ISlider
            IScrollBar
            ISpinButton
            ITextControl
               IButton
                  IPushButton
                  ISettingButton
                     I3StateCheckBox
                     ICheckBox
                     IRadioButton
               IEntryField
                  IComboBox
               IGroupBox
               IMultiLineEdit
               IStaticText
                  IBitmapControl
                     IIconControl
                  IInfoArea
               ITitle
         IFrameWindow
            IFileDialog
            IFontDialog
         IHelpWindow
         IMenu
            IMenuBar
            IPopUpMenu
            ISubMenu
            ISystemMenu
         IObjectWindow
```

Handler Classes

```
IBase
   IVBase
      IHandler
         ICnrDrawHandler
         ICnrEditHandler
         ICnrHandler
         ICommandHandler
         IDDEClientConversation
         IDDETopicServer
         IEditHandler
         IFileDialogHandler
         IFocusHandler
         IFontDialogHandler
         IFrameHandler
         IHelpHandler
         IKeyboardHandler
         IListBoxDrawItemHandler
         IMenuDrawItemHandler
         IMenuHandler
            ICnrMenuHandler
            IInfoArea
         IMouseClickHandler
         IPageHandler
         IPaintHandler
         IResizeHandler
         IScrollHandler
         ISelectHandler
         IShowListHandler
         ISliderDrawHandler
         ISpinHandler
```

Event Classes

```
IBase
   IVBase
      IEvent
         ICnrDrawBackgroundEvent
         ICommandEvent
         IControlEvent
            ICnrEvent
               ICnrEditEvent
                  ICnrBeginEditEvent
                  ICnrEndEditEvent
                  ICnrReallocStringEvent
               ICnrEmphasisEvent
               ICnrEnterEvent
               ICnrHelpEvent
               ICnrQueryDeltaEvent
               ICnrScrollEvent
            IDrawItemEvent
               ICnrDrawItemEvent
               IListBoxDrawItemEvent
               IMenuDrawItemEvent
               INotebookDrawItemEvent
            IPageEvent
               IPageHelpEvent
               IPageRemoveEvent
               IPageSelectEvent
         IFileDialogEvent
         IFrameEvent
            IFrameFormatEvent
         IHelpErrorEvent
         IHelpHyperTextEvent
         IHelpMenuBarEvent
         IHelpNotifyEvent
         IHelpSubItemNotFoundEvent
         IHelpTutorialEvent
         IKeyboardEvent
         IMenuEvent
         IMouseClickEvent
         IPaintEvent
         IResizeEvent
         IScrollEvent
      IEventData
         IEventParameter1
         IEventParameter2
         IEventResult
```

Event Classes - DDE Events

```
IBase
   IVBase
      IEvent
         IDDEBeginEvent
         IDDEEndEvent
            IDDEClientEndEvent
         IDDEEvent
            IDDEAcknowledgeEvent
               IDDEClientAcknowledgeEvent
                  IDDEAcknowledgePokeEvent
                  IDDEAcknowledgeExecuteEvent
               IDDEServerAcknowledgeEvent
            IDDESetAcknowledgeInfoEvent
               IDDEClientHotLinkEvent
               IDDEDataEvent
               IDDEExecuteEvent
               IDDEPokeEvent
               IDDERequestDataEvent
               IDDEServerHotLinkEvent
```

Data Types and Attributes Classes

```
IBase
    IColor
        IDeviceColor
        IGUIColor
    IDate
    IHandle
        IAccelTblHandle
        IAnchorBlockHandle
        IBitmapHandle
            ISystemBitmapHandle
        IEnumHandle
        IMessageQueueHandle
        IModuleHandle
        IPageHandle
        IPointerHandle
            ISystemPointerHandle
        IPresSpaceHandle
        IProcessId
        IProfileHandle
        ISemaphoreHandle
        IStringHandle
        IThreadId
        IWindowHandle
    IPair
        IPoint
        IRange
        ISize
    IRectangle
    IString
        I String
    ITime
    IVBase
        IBuffer
            IDBCSBuffer
        IFont
        IStringTest
            IStringTestMemberFn

IStringEnum
```

Settings and Styles Classes

```
IBase
   IBitFlag
      I3StateCheckBox::Style
      IBitmapControl::Style
      IButton::Style
      ICanvas::Style
      ICheckBox::Style
      IComboBox::Style
      IContainerControl::Attribute
      IContainerControl::Style
      IControl::Style
      IEntryField::Style
      IFileDialog::Style
      IFontDialog::Style
      IFrameWindow::Style
      IGroupBox::Style
      IIconControl::Style
      IListBox::Style
      IListBoxDrawItemHandler::DrawFlag
      IMenuDrawItemHandler::DrawFlag
      IMenuItem::Attribute
      IMenuItem::Style
      IMessageBox::Style
      IMultiLineEdit::Style
      INotebook::PageSettings::Attribute
      INotebook::Style
      IOutlineBox::Style
      IProgressIndicator::Style
      IPushButton::Style
      IRadioButton::Style
      IScrollBar::Style
      ISetCanvas::Style
      ISlider::Style
      ISpinButton::Style
      ISplitCanvas::Style
      IStaticText::Style
      IViewPort::Style
      IWindow::Style
   IFileDialog::Settings
   IFontDialog::Settings
   IHelpWindow::Settings
   IVBase
      INotebook::PageSettings
```

Support Classes

```
IBase
    IAccelerator
    IDDEActiveServer
    IFrameExtension
    IMenuItem
    ISWP
    ISWPArray
    IVBase
        IComboBox::Cursor
        IContainerColumn
        IContainerControl::ColumnCursor
        IContainerControl::CompareFn
        IContainerControl::FilterFn
        IContainerControl::Iterator
        IContainerControl::ObjectCursor
        IContainerControl::TextCursor
        IContainerObject
        IListBox::Cursor
        INotebook::Cursor
        IProfile::Cursor
        ISpinButton::Cursor
        IWindow::ChildCursor

ISequence<>
    IFrameExtensions

ISet<>
    IDDEActiveServerSet
    IDDEClientHotLinkSet
```

Exception and Error Handling Classes

```
IBase
    IVBase
        IErrorInfo
            IGUIErrorInfo
            ISystemErrorInfo
        ITrace
        IWindow::ExceptionFn

IException
    IAccessError
    IAssertionFailure
    IDeviceError
    IInvalidParameter
    IInvalidRequest
    IResourceExhausted
        IOutOfMemory
        IOutOfSystemResource
        IOutOfWindowResource

IException::TraceFn

IExceptionLocation

IMessageText
```

Appendix B. Class Library Conventions

The purpose of this appendix is to introduce you to the conventions used in the User Interface Class Library. These conventions are:

- File names
- Class, function, and data member names
- Enumerations
- Function return types
- Function arguments
- Other standards

File Names

All files provided by the User Interface Class Library begin with the letter "I," such as IAPP.HPP. File names have a maximum of eight characters, including the "I." Following is a list of the file name extensions that are used:

Ixxxxxxx.HPP User Interface Class Library header file.

Ixxxxxxx.INL User Interface Class Library inline functions.

DDE4MUII.LIB The multi-threaded user interface import library.

DDE4MUI.DLL The multi-threaded user interface import dynamic-link library.

DDE4MUI.DEF The multi-threaded user interface import module definition file.

DDE4MUI.RSP The multi-threaded user interface import linker automatic response file.

DDE4MUIB.LIB The multi-threaded user interface regular object library (base).

DDE4MUIC.LIB The multi-threaded user interface regular object library (controls).

DDE4MUID.LIB The multi-threaded user interface regular object library (DDE).

© Copyright IBM Corp. 1993

The file name generally indicates the class or classes it contains. For example, the IAPP.HPP file contains the `IApplication` and `ICurrentApplication` classes. Refer to the *IBM C/C++ Tools: User Interface Class Library Reference* for an appendix that contains cross-reference tables for the header files and the classes they contain.

Class Names, Function Names, and Data Member Names

Class names are mixed case, with the first letter of each word capitalized, as in `ICurrentApplication`. All class names in the global name space begin with the letter "I."

Function names and data member names are also mixed case, except the first letter is always lower case, as in the `autoSize` data member. Here are some more general rules about class and function names:

- Acronyms are upper case, as in `IDBCSBuffer`. DBCS is the acronym for double-byte character set. Other acronyms you will see are GUI (graphical user interface) and DDE (dynamic data exchange).
- Abbreviations are mixed case, as `IPresSpaceHandle`, which is the class for presentation space handles.
- Functions that query begin with a prefix that implies a query is being conducted, such as "is" or "has." The `IDragItem` class, for example, has the `isCopyable` function, which queries whether an object can be copied.
- Functions that render an object as a different type begin with the "as" prefix, as in `asUnsignedLong`, which renders an object as an unsigned long.
- Functions that provide enabling or disabling capabilities begin with the "enable" or "disable" prefix, respectively. The `IEntryField` class, for example, provides the `enableAutoScroll` function, which enables automatic scrolling.
- Functions that set something begin with the "set" prefix. The `setDefaultStyle` function, to follow the preceding example, is used to set the default style for a class.

- Functions that get something have no “get” prefix. For example, the `defaultStyle` function is used by many classes to get the default style for that class.
- Functions that act on objects are verbs, such as `copy`, `move`, and so forth.
- Function names and arguments are written to be virtually self explanatory. The following example would move the `IWindow` object `aWindow` to the position specified by the `IPoint` object `aPoint`.

  ```
  aWindow.moveTo( aPoint );
  ```

- Many functions that toggle the state of an object are provided with an optional Boolean argument that can be used to perform the opposite action of the function. This allows the result of a prior query function to be used as an input argument, such as:

  ```
  Boolean initialVisibility = isVisible();
  hide();
  /* Do some hidden work */
  show(initialVisibility);
  ```

Enumerations

Here are the conventions followed for enumeration types and enumerators:

- The first character of each enumeration name is upper case. If two words are joined, each begins with an upper case letter.
- Enumerators use the same naming conventions as functions; they begin with lower case letters, but if two words are joined, the second begins with an upper case letter.

Function Return Types

Here are the return types for the various types of functions:

- A testing function typically returns a Boolean (true or false):

  ```
  Boolean isValid() const
  ```

- Other accessor functions typically return an object:

  ```
  ISize size() const;              //Returns an object
  IWindow* static owner();          //Returns a pointer to an object
  static IWindow* desktopWindow();   //Returns a pointer that points to an
                                  //object
  ```

- Functions that act on an object return an object reference:

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

  This allows the chaining of function calls:

  ```
  aWindow.moveTo(IPoint(1 ,1 )).show();
  ```

Function Arguments

Function arguments are usually passed using the following conventions:

- Built-in types (ints, doubles) and enumerations are passed in by value.
- Objects are passed by reference (a `const` reference if the argument is not modified by the function).
- "Optional" objects are passed by pointer. For example, a 0 pointer can be passed.
- `IWindow` objects are usually passed by pointer.
- `IContainerObject`s are usually passed by pointer.
- "Strings" are passed as `const char *`. This enables you to pass either an `IString` or a literal character array.

Other Standards

The following are miscellaneous standards followed by the User Interface Class Library:

- Header files are wrapped to ensure that files are not included more than once.
- All functions that can be inlined are placed in separate INL files with a user option (I_NO_INLINES) to determine whether they should be inlined into the application code. If you do not want to inline these functions, then define I_NO_INLINE.
- ISYNONYM.HPP contains the names of the types and values which are in the global name space but do not begin with the letter "I." If you have collisions with other libraries, the names in ISYNONYM.HPP can be changed.

Bibliography

This bibliography lists the publications comprising the IBM C/C++ Tools library and publications of related IBM products referenced in this book. The list of related publications is not exhaustive but should be adequate for most C/C++ Tools users.

The IBM C/C++ Tools Library

The following books are part of the IBM C/C++ Tools library.

- *Programming Guide*
- *Migration Guide*
- *Reference Summary*
- *Debugger Introduction*
- *Execution Trace Analyser Introduction*
- *Browser Introduction*
- *C/C++ Tools Installation*
- *C Library Reference*
- *C Language Reference*
- *C++ Language Reference*
- *Standard Class Library Reference*
- *User Interface Class Library Reference*
- *Collection Class Library Reference*

C and C++ Related Publications

- *SAA Common Programming Interface C Reference*, SC09-1308
- *Portability Guide for IBM C*, SC09-1405
- *American National Standard for Information Systems — Programming Language C (X3.159-1989)*
- *International Standard C ISO/IEC 9899:1990(E)*
- *Draft Proposed American National Standard for Information Systems — Programming Language C++ (X3J16/92-0060)*

IBM WorkFrame/2 Publications

- *IBM WorkFrame/2: Introduction*, S10G-4475

IBM OS/2 2.0 Publications

The following books describe the OS/2 2.0 operating system and the Developer's Toolkit.

- *IBM OS/2 2.0 Overview Manual*, S84F-8465
- *IBM OS/2 2.0 Installation Guide*, S84F-8464
- *IBM OS/2 2.0 Quick Reference*, S10G-5964
- *Getting Started*, S10G-6199

© Copyright IBM Corp. 1993

IBM OS/2 2.0 Technical Library

The following books make up the OS/2 2.0 Technical Library (10G3356).

- *Application Design Guide*, S10G-6260
- *Programming Guide*, S10G-6261
- *Information Presentation Facility Guide and Reference*, S10G-6262
- *System Object Model Guide and Reference*, S10G-6309
- *Control Program Programming Reference*, S10G-6263
- *Presentation Manager Programming Reference Volume 1*, S10G-6264
- *Presentation Manager Programming Reference Volume 2*, S10G-6265
- *Presentation Manager Programming Reference Volume 3*, S10G-6272
- *Physical Device Driver Reference*, S10G-6266
- *Virtual Device Driver Reference*, S10G-6310,
- *Presentation Manager Driver Reference*, S10G-6267
- *Procedures Language 2/REXX Reference*, S10G-6268,
- *Procedures Language 2/REXX User's Guide*, S10G-6269
- *SAA Common User Access Guide to User Interface Design*, SC34-4289
- *SAA Common User Access Advanced User Interface Design Guide*. SC34-4290

Other Books You Might Need

The following list contains the titles of IBM books that you might find helpful. These books are not part of the C/C++ Tools, WorkFrame/2, or OS/2 2.0 libraries.

BookManager* READ/2 Publications

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

Systems Application Architecture* Publications

- *An Overview*, GC26-4341
- *C Reference Level 1*, SC26-4353
- *C Reference Level 2*, SC09-1308
- *Common User Access: Panel Design and User Interaction*, SC26-4351
- *Communications Reference*, SC26-4399
- *Database Reference*, SC26-4353
- *Dialog Reference*, SC26-4356
- *SAA Common Programming Interface PL/I Reference*, SC26-4381
- *Presentation Reference*, SC26-4359
- *Procedures Language Reference*, SC26-4358
- *Query Reference*, SC26-4349
- *Writing Applications: A Design Guide*, SC26-4362

Glossary

This glossary defines terms and abbreviations that are used in this book. It does not include all terms previously established in the *SAA CPI C Reference - Level 2*. If you do not find the term you are looking for, refer to the *IBM Dictionary of Computing*, SC20-1699.

This glossary includes terms and definitions from the *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018.

A

abstract class. A class with at least one pure virtual function that is used as a base class for other classes. The abstract class represents a concept; classes derived from it represent implementations of the concept. You cannot have a direct object of an abstract class. See also *base class*.

abstraction (data). See *data abstraction*.

access. An attribute that determines whether or not a class member is accessible in an expression or declaration.

address. A name, label, or number identifying a location in storage, a device in a system or network, or any other data source.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

Note: IBM has defined an extension to ASCII code (characters 128-255).

API. Application program interface.

application. The use to which an information processing system is put, for example, a payroll application, an airline reservation application, a network application.

application program interface (API). The formally defined programming language interface between an IBM system control program or a licensed program and the user of the program.

argument. In a function call, an expression that represents a value that the calling function passes to the function specified in the call. Also called a parameter.

array. A variable that contains an ordered group of data objects. All objects in an array have the same data type.

ASCII. American National Standard Code for Information Interchange.

asynchronous (ASYNC). Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions.

© Copyright IBM Corp. 1993

B

base class. A class from which other classes are derived. A base class may itself be derived from another base class. See also *abstract class*.

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.

bit. A binary digit.

block. The unit of data transmitted to and from a device. Each block contains one record, part of a record, or several records.

buffer. A portion of storage used to hold input or output data temporarily.

byte. For IBM C compilers, 8 bits equal 1 byte.

C

call. To transfer control to a procedure, program, routine, or subroutine.

character set. A group of characters used for a specific reason; for example, the set of characters a printer can print or a keyboard can support.

class. A C++ aggregate that may contain functions, types, and user-defined operators in addition to data. Classes may be defined hierarchically, allowing one class to be an expansion of another, and may restrict access to its members.

class library. A collection of C++ classes.

Collection Classes. A set of classes that provide basic functions and can be used as base classes.

command. A request to perform an operation or run a program. associated with a command, the resulting character string is a single command.

compiler. A program that translates instructions written in a programming language (such as C language) into machine language.

condition. A relational expression in a program or procedure that can be evaluated to a value of either true or false.

const. An attribute of a data object that declares the object cannot be changed.

constructor. A special class member function that has the same name as the class and is used to construct and possibly initialize class objects.

conversion. A change in the type of a value. The compiler converts both values to a common form before adding the values. Because accuracy of data representation varies among different data types, information may be lost in a conversion.

copy constructor. A constructor used to make a copy of a class object from another class object of the same class type.

cursored emphasis. When the selection cursor is on a choice, that choice has cursored emphasis.

D

data abstraction. A data type with a private representation and a public set of operations. The C++ language uses the concept of classes to implement data abstraction.

data object. A storage area used to hold a value.

data stream. A continuous stream of data elements being transmitted, or intended for

transmission, in character or binary-digit form, using a defined format.

DBCS. (1) See *double-byte character set*. (2) See *ASCII*.

declaration. A description that makes an external object or function available to a function or a block.

declare. To identify the variable symbols to be used at preassembly time.

default. An attribute, value or option that is used when no alternative is specified by the programmer.

default constructor. A constructor that takes no arguments, or for which all the arguments have default values.

definition. A data description that reserves storage and may provide an initial value.

delete. A C++ keyword that identifies a free storage deallocation operator.

derived class. A class that inherits from a base class. You can add new data members and member functions to the derived class. A derived class object can be manipulated as if it were a base class object. The derived class can override virtual functions of the base class.

destructor. A special member function that has the same name as its class, preceded by a tilde (~), and that "cleans up" after an object of that class, for example, freeing storage that was allocated when the object was created. A destructor has no arguments and no return type.

directory. A file containing the names and controlling information for other files or other directories.

double-byte character set (DBCS). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets.

Because each character requires 2 bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that are DBCS capable.

dynamic. Pertaining to an operation that occurs at the time it is needed rather than at a predetermined or fixed time.

E

EBCDIC. See *extended binary-coded decimal interchange code*.

element. A data object in an array.

exception. (1) Under the OS/2 operating system, a user or system error detected by the system and passed to an OS/2 or user exception handler. (2) For C++, any user, logic, or system error detected by a function that does not itself deal with the error but passes the error on to a handling routine (also called throwing the exception).

exception handling. A type of error handling that allows control and information to be passed to an exception handler when an exception occurs. Under the OS/2 operating system, exceptions are generated by the system and handled by user code. In C++, `try, catch`, and `throw` expressions are the constructs used to implement C++ exception handling.

expression. A representation for a value. For example, variables and constants appearing alone or in combination with operators.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

extension. (1) An element or function not included in the standard language. (2) File name extension.

F

file. A collection of data that is stored and retrieved by an assigned name.

file handle. A value created by the system that identifies a drive, directory, and file so that the file can be found and opened.

file name. The name used to identify a file.

friend function. A function that is granted access to the private and protected parts of a class. It is named in the declaration of the other class with the prefix `friend`.

function. A named group of statements that can be invoked and evaluated and can return a value to the calling statement.

G

global. Pertaining to information available to more than one program or subroutine.

H

header file. A file that contains system-defined control information that precedes user data.

I

identifier. A sequence of letters, digits and underscores used to designate a data object or function.

inheritance. An object-oriented programming technique that allows you to use existing classes as bases for creating other classes.

initialize. To set the starting value of a data object.

input. Data to be processed.

instance. Synonym for object, a particular instantiation of a data type.

interrupt. A temporary suspension of a process caused by an external event, performed in such a way that the process can be resumed.

L

label. (1) An identifier followed by a colon. It is the target of a goto statement. (2) An identifier within or attached to a set of data elements.

library. (1) A collection of functions, function calls, subroutines, or other data. (2) A set of object modules that can be specified in a link command.

link. To interconnect items of data or portions of one or more computer programs; for example, linking of object programs by a linkage editor to produce an executable file.

linker. A program that resolves cross-references between separately compiled object modules and then assigns final addresses to create a single executable program. load module.

list box. A control window containing a vertical list of selectable description.

M

macro. An identifier followed by arguments (may be a parenthesized list of arguments) that the preprocessor replaces with the replacement code located in a preprocessor `#define` directive.

main function. A function with the identifier `main` that is the first user function to get control when program execution begins. Each C program must have exactly one function named `main`.

map. A set of values having a defined correspondence with the quantities or values of another set.

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters.

member. (1) A data object in a structure or a union. (2) In C++, classes and structures can also contain functions and types as members.

member function. An operator or function that is declared as a member of a class. A member function has access to the private and protected data members and member functions of objects of its class.

method. Synonym for member function.

module. A program unit that usually performs a particular function or related functions, and that is distinct and identifiable with respect to compiling, combining with other units, and loading.

N

nested class. A class defined within the scope of another class.

NULL. A pointer guaranteed not to point to a data object.

null character (\0). The ASCII or EBCDIC character with the hex value (all bits turned off).

O

operand. An entity on which an operation is performed.

operating system. Software that controls functions such as resource allocation, scheduling, input/output control, and data management.

operation. A specific action such as add, multiply, shift.

operator. A symbol (such as +, −, *) that represents an operation (in this case, addition, subtraction, multiplication).

OS/2. Pertaining to the operating system for the PS/2 workstation.

overflow. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

overlay. To write over existing data in storage.

P

pad. To fill unused positions in a field with data, usually zeros, ones, or blanks.

pointer. A variable that holds the address of a data object or function.

private. Pertaining to a class member that is only accessible to member functions and friends of that class.

process. An instance of an executing application and the resources it uses.

program. One or more files containing a set of instructions conforming to a particular programming language syntax.

protected. Pertaining to a class member that is only accessible to member functions and friends of that class, or to member functions and friends of classes derived from that class.

public. Pertaining to a class member that is accessible to all functions.

R

record. The unit of data transmitted to and from a program.

register. A storage area commonly associated with fast-access storage, capable of storing a specified amount of data such as a bit or an address.

run. To cause a program, utility, or other machine function to be performed.

S

scalar. An arithmetic object, or a pointer to an object of any type.

scope. That part of a source program in which an object is defined and recognized.

semaphore. An object used by multithread applications for signalling purposes and for controlling access to serially reusable resources.

source file. A file that contains source statements for such items as language programs and data description specifications.

stack. An area of storage used for keeping variables associated with each call to a function or block.

statement. An instruction that ends with the character **;** (semicolon) or several instructions that are surrounded by the characters { and }.

static. (1) Pertaining to properties that can be established before execution of a program, for example, the length of a fixed length variable. (2) Pertaining to an operation that occurs at a predetermined or fixed time. (3) Pertaining to a variable that receives private and permanent storage, and is not known outside of the block or file in which it is declared.

stream. See *data stream*.

structure. A construct that contains an ordered group of data objects. Unlike an array, the data objects within a structure can have varied data types.

system default. A default value defined in the system profile.

T

tag. One or more characters attached to a set of data that identifies the set.

task. One or more sequences of instructions treated by a control program as an element of work to be accomplished by a computer.

template. A family of classes or functions with variable types.

this. A C++ keyword that identifies a special type of pointer in a member function, that references the class object with which the member function was invoked.

thread. A unit of execution within a process.

type. See *data type*.

U

union. A construct that can hold any one of several data types, but only one data type at a time.

V

virtual function. A function of a class that is declared with the keyword `virtual`. The implementation that is executed when you make a call to a virtual function depends on the type of the object for which it is called, which is determined at run time.

W

whitespace. Space characters, tab characters, form feed characters, and new-line characters.

Index

A

accelerator keys
  adding to the application 207
adding a resource file
  example 155
application classes
  critical sections 122
  description 17
  hierarchy 259
  overview 4
  protecting data 121
  threads 116
  tracing 109
attribute classes
  overview 5

C

canvas classes
  creating 217
  DBCS/NLS usage 133
  description 44
  multicell canvas 51
  set canvas 47
  split canvas 45
  viewport 54
cascaded menu
  adding to a pull-down menu 207
character data
  managing with IString class 69
check box control
  description 36
  events 64
  handlers 64
class library application
  files used to create 141
class library application *(continued)*
  structure 141
class names
  coding conventions 270
client window
  example 242
clipboard
  multiple-line entry control 85
coding conventions
  description 269
color
  setting in a window 33
combo box control
  events 64
  handlers 64
command line parameters
  setting 17
constants definition file
  example 161
container control
  adding and removing objects 89
  columns 97
  creating objects 87
  cursors 93
  description 86
  details view 97
  events 64
  filter objects 90
  handlers 64
  make popup menu event 64
  pop-up menu 100
  views 95
contextual help
  description xii
controls
  check box control 36
  container control 86

© Copyright IBM Corp. 1993

controls *(continued)*
entry field control 33
information area control 28
multiple-line entry field control 81
notebook text control 101
push button control 34
radio button control 38
slider control 40
static text control 31
creating a main window
example 145
Hello World version 1 148
creating a static text control
example 167
Hello World version 1 149
creating simple dialogs
example 189
creating the main window
Hello World version 2 164
critical sections
description 122
cursors
container control 93
description 58
multiple-line entry control 83
sample code 59

D

data member names
coding conventions 270
data type classes
overview 5
DBCS
description 133
def files
description 269
dialogs
creating simple 189
standard dialogs 123
directory location xii
DLL
resource 133

E

entry field control
description 33
events 64
handlers 64
sample code 34
styles example 56
enumerations
coding conventions 271
event
command 64
control 64
description 65
enter 64
extending 105
focus 64
keyboard 64, 133
make popup menu 64
menu 129
menu showing 64
paint window 64
resize 64
selected 64
summary table 65
system command 64
event classes
hierarchy 262
overview 4
event handling
example 173
examples
directory location xii
exception classes
exception handling 112
hierarchy 267

exception handling
  default exception handler 114
  description 112
exiting an application 18

F

file dialog
  description 123
files
  class library conventions 269
  def files 269
  DLL resources 18
  hpp files 269
  inl files 269, 272
  lib files 269
  string resources 18
  user resource 19
font class
  description 77
  sample code 78
font dialog
  description 125
frame extensions
  description 21
  information area 28
  menu bar 24
  minimized icon 24
  status area 30
  title bar 23
frame window
  styles 22
function arguments
  coding conventions 272
function member names
  coding conventions 270
function return types
  coding conventions 271

H

handler
  command 64
  container menu 64
  description 62
  edit 64
  focus 64
  keyboard 64
  menu 64, 128
  paint 64
  resize 64
  select 64
  summary table 63
  writing a handler 67
handler classes
  hierarchy 261
  overview 4
hardware requirements xi
Hello World sample application
  version 1 145
  version 2 155
  version 3 173
  version 4 189
  version 5 217
  version 6 251
Hello World version 1
  compiling and linking the application 152
  creating a static text control 149
  creating the main window 148
  files 147
  running the application 151
  tasks performed 148
Hello World version 2
  compiling and linking 171
  creating the main window 164
  files 158
  tasks performed 164
Hello World version 3
  compiling and linking 187
  files 175

Hello World version 3 *(continued)*
tasks performed 182
Hello World version 4
compiling and linking 215
files 191
tasks performed 205
Hello World version 5
compiling and linking 249
files 219
tasks 240
Hello World version 6
compiling and linking 256
files 254
tasks 255
help
contextual xii
creating for your application 217
description 130
push button example 36
setting up 243
hpp files
description 269

I

icon file
example 162
information area
creating 170
description 28
setting up 247
inl files
description 269, 272

L

lib files
description 269
list box control
cursor sample code 59
events 64
list box control *(continued)*
handlers 64
styles examples 56
styles within a canvas 45

M

main window
setting the size 171
menu bar
creating 185
description 24
example of modifying 206
menus
events 64
handlers 64
help menu 131
menu bar 24
menu showing event 64
pop-up menu 128
system menu 21
message box
description 127
minimized icon
description 24
multicell canvas
description 51
multiple-line entry control
clipboard 85
cursors 83
description 81
events 64
handlers 64
interface to files 83

N

NLS
description 133
example 251

notebook control
description 101
page settings 104
styles 102

O

operating system requirements xi

P

pop-up menu
container control 100
description 128
protecting data
description 121
pull-down menu
adding a cascaded menu 207
example 206
push button control
adding in a set canvas 211
description 34
events 64
example of creating 213
handlers 64
sample code 213
setting text 214
styles example 35

R

radio button control
description 38
events 64
handlers 64
rc file
setting a text string 169
requirements
hardware xi
installation xii
operating system xi
requirements *(continued)*
software xi
resource file
adding 155
description 18
example 162
running an application 18

S

set canvas
adding push buttons 211
description 47
example of creating 211
setting classes
hierarchy 265
overview 5
setting up help
example 243
setting up the client window
example 242
setting up the information area
example 247
setting up the status area
example 248
shortcut keys
adding to the application 207
slider control
description 40
events 64
handlers 64
software requirements xi
spin button control
handlers 64
split canvas
description 45
standard dialogs
description 123
file dialog 123
font dialog 125

static text control
alignment styles 32
description 31
example 149
sample code 31
status area
description 30
setting up 248
specifying location and height 184
string class
accessors 70
comparison operators 72
converting strings 74
DBCS/NLS 133
managing character data 69
manipulating text 76
modifying and aligning strings 75
reading and writing text 69
testing 71
string resources
descriptions 18
style classes
hierarchy 265
overview 5
styles
container control 87
description 55
frame window 22
multiple-line entry field control 81
notebook control 102
push button control 35
push button example 36
sample code 57
slider example 42
static text control 32
support classes
hierarchy 266
overview 5

T

text
aligning in a window 32
setting in a window 32
text string
setting from an RC file 169
threads
description 116
title bar
description 23
tracing
description 109

U

User Interface Class Library
running and exiting an application 18
user resource files 19
user resource files 19
user-created control
creating 217

V

viewport
description 54

W

window classes
cursors 58
events 64
handlers 64
help 130
hierarchy 260
message box 127
overview 4
pop-up menu 128
standard dialogs 123
styles 55

windows
defining layout with canvas classes 44
sizing with canvas classes 44