VisualAge® C++ Professional for AIX®

IBM® Open Class™: Overview

Version 5.0

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

Edition Notice

This edition applies to Version 5.0 of IBM VisualAge C++ and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Contents

Notices

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

Programming Interface Information

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

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

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

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

Trademarks and Service Marks

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

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

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

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

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

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

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

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

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

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

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

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

Industry Standards

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

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

About This Book

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

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

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

Chapter 1. IBM Open Class Overview

The IBM Open Class (IOC) is a comprehensive library of C++ classes provided with VisualAge C++ that you can use to develop applications.

The IBM Open Class takes advantage of the data abstraction and object-oriented paradigms adopted by C++. This library simplifies otherwise tedious, convoluted, or error-prone tasks through the use of straightforward interfaces. For example, the user interface classes enable you to create applications for windowed environments such as Motif, OS/2's Presentation Manager®, or Windows®, without the necessity of learning obscure system calls.

Because this library consists of C++ classes, you can create customized classes through the C++ language's mechanism of inheritance.

The IBM Open Class supports AIX, Windows, and OS/2®. Consequently, you can port your IBM Open Class applications between platforms with minimal effort.

This documentation assumes that you are proficient with both the C++ programming language, and with object-oriented programming concepts and techniques. Refer to Bjarne Stroustrup's *The C++ Programming Language: Third Edition* by Bjarne Stroustrup for detailed information on this language. This reference is included with the VisualAge C++ PDF files.

The Organization of the IBM Open Class
The IBM Open Class is organized into several broad categories as follows:

- Application Control
- Streams
- File Systems
- Text and Internationalization
- 2D Graphics
- User Interface
  - WIN OS/2 Multimedia
  - WIN OS/2 Dynamic Data Exchange
- Error Handling, Tracing, and Testing

Three other categories have comparable functionality to the ANSI C++ Standard Library:

- Collections
- Math
- Non-ANSI Input/Output Streaming

The ANSI C++ Standard Library provides classes and data structures that efficiently implement the functionality of Collections and Math classes. We suggest that you use the ANSI C++ Standard Library instead of the IBM Open Class for new applications that need this functionality. Refer to Stroustrup's *The C++ Programming Language* for more information on the ANSI C++ Standard Library.

Application Control
This category provides support for multi-threaded execution environments:

- Process classes create and manipulate external processes.
- Thread classes create and manipulate threads.
- Notification classes notify interested objects of changes in other objects.
- Reference counting classes manage thread safe access to multiply-referenced objects.

Other classes in this category interact with and control the application and its modules, libraries, resources, environments, profile, and timers.

Object-Persistent Streaming
This category implements persistent storage mechanisms for Open Class Library components. It includes the streaming classes that support streaming data in C++ objects in and out of persistent storage.

File Systems
This category provides portable abstractions that allow you to manipulate physical file system entities such as volumes, directories, and files.

Text and Internationalization
This category provides support for Unicode text strings and easily localizable components. This category includes the following groups of classes:

- Unicode support classes inquire about the stylistic and semantic properties of characters, character sets, and scripts (writing systems). These classes also store and manipulate Unicode text styles.
- Internationalization classes create international applications and manipulate international text. This includes language-sensitive comparison of text strings, conversion between character sets, and a locale mechanism for access to portable and host-specific resources.

2D Graphics
This category supports the platform-independent creation, manipulation, and rendering of 2D graphic objects. This category includes these class groups:

- Geometry classes define forms for path, area, and image geometries.
- Attributes describe graphic characteristics such as paint, pen, and joints, so you can create groups of attributes to apply to graphic objects that you are rendering.
- Transformations let you apply mathematical transformations such as scaling, rotating, and translating to 2D graphic objects.
- Modeling classes combine geometry objects with attributes and transformations for rendering and storing graphics and creating hierarchies of graphics.
- Rendering pipeline classes implement the rendering of 2D graphic objects.

User Interface
This category provides support for building the graphical user interface of an application:

- Window, menu, handler, and event classes encapsulate the basic graphical building blocks that are used to construct application windows. This encapsulation separates window position and appearance (windows and menus) from event handling (handlers and events).
- Standard control classes support standard controls such as entry field, static text, and buttons.
- Advanced control, dialog, and handler classes support controls such as containers, notebooks, toolbars, and font and file dialogs.
- Direct manipulation classes support the “drag-and-drop” mechanism.

Multimedia

WIN OS/2 The Multimedia classes support the creation of applications that integrate text and graphics with a combination of audio, motion video, images, and animation. You can implement an interface for your application that looks like the controls of common electronic devices, such as stereo components and video cassette recorders (VCRs).

Your application can use these controls as interfaces to control audio and video media that is presented to the user. Support for media devices is abstracted into classes that contain the data and functions essential for the operation of the real-world devices that they model. These real-world devices include the following:

- Master audio
- Audio amplifier-mixer
- CD audio player
- CD Extended-Architecture player
- Waveform audio player
- MIDI sequencer
- Digital video player

Dynamic Data Exchange

WIN OS/2 *Dynamic data exchange* (DDE) is a client/server protocol for communicating between two applications running on the same machine. *Client/server* implies a relationship between two applications where the *client* sends requests to the *server*. The server handles the requests and provides services to the client application which then consumes those services. The server-provided services can be either of the following:

- Data
- The ability to execute commands on behalf of a client

As a client/server protocol, DDE enables data to be dynamically exchanged, and thus shared, between two applications running on the same machine. Applications are shielded from the operating details and can share data once they agree on the type of data being exchanged. DDE applications can exchange data on an ongoing basis without user intervention.

Error Handling, Tracing, and Testing

This category provides support for building robust and well-behaved applications:

- Exception classes detect and convey information about unusual circumstances in applications.
- Tracing classes provide support for your tracing code.
- Test classes help you create and run unit tests for your application.

Collections

This category provides a set of commonly used abstract data types including sets, maps, sequences, trees, stacks, and queues. You can use the notification classes from the application control classes to allow observation of changes within a collection.

Math

This category allows you to manipulate complex numbers. It also provides cross-platform 64-bit support.

Non-ANSI Input/Output Streaming

The non-ANSI Input/Output Streaming Library, also called the Unix Systems

Laboratories (USL) or the AT&T I/O Stream Library, provide the standard input and output capabilities for C++. In C++, input and output are described in terms of *streams*. The processing of these streams is done at two levels:

1. The first level treats the data as sequences of characters.
2. The second level treats the data as a series of values of a particular type.

The I/O Stream Classes predefine streams for standard input, standard output, and standard error.

The ANSI C++ Standard Library
The ANSI C++ Standard library, which is comprised of the Standard Template Library (STL) and the Standard C++ Library, is provided with VisualAge C++. The IBM Open Class contains some classes that duplicate functions provided by the ANSI C++ Standard library.

We recommend that you take advantage of the ANSI C++ Standard Library in your applications:

- Use the STL's containers, iterators, and algorithms instead of the IOC Collections
- Use the STL's Numerics Library instead of the IOC Math classes
- Use the ANSI C++ I/O Stream classes instead of the non-ANSI I/O Stream classes.

This release of IBM Open Class uses the ANSI C++ I/O Stream classes and container class templates. The non-ANSI I/O Stream Library will continue to be shipped with the product and may be deprecated in a future release. You may use either the ANSI or USL stream library with the IBM Open Class by defining or undefining the macro __IOC_ANSI_STREAM.

The IOC Collections will continue to be shipped with the product. Some of the user interface classes are dependent on the function provided in these classes (for example, support for notification).

RELATED REFERENCES
VisualAge C++ PDF Files

IBM Open Class Applications

Include the Open Class Library Headers
A *class library* is a collection of classes with well-defined interfaces and operations. To use these classes, you have to first make these interfaces visible to your program. This is usually done by including the appropriate header files in your programs.

To include an interface, use the directive #include <filename>, where filename is the name of the header file. Place this statement at the beginning of the program that requires any of the classes, function, or operators defined in the header file. Then, in the body of your program, you can use a class, function, or operator defined in the header file, as well as derive new classes and overload the functions and operators.

For the incremental compiler, you can also list the header files in the configuration file instead of your source file. A *configuration file* is a set of source files, input libraries, and processing options that are used to generate one or more targets. A VisualAge C++ project must have a configuration file.
Create Applications and Naming Files

Creating an Open Class Library application is the same as creating a C++ application. You begin by designing the interfaces of your classes and make use of any existing Open Class Library classes as much as possible. Although there is no restriction on how C++ files are named, it is often a good idea to use a particular convention. For an Open Class Library application, the source file is usually named after the name of the class that resides in the file. For example, a class "ITest" is defined in the file "itest.cpp" and its interface is defined in "itest.hpp".

The following list describes files that a typical Open Class application requires:

File Name	Contains
filename.cpp	Primary C++ code for your application.
filename.icc	The configuration file of your application.
filename.hpp	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.
filename.h	A header file that contains your symbolic definitions. Include these as macro source files in your project's configuration.
i*xxxxxxx*.hpp	A header file that contains information about an Open Class Library class that your application uses. All Open Class Library header files begin with the letter "i."
makefile	The make file of your application

Use Command-Line Arguments
With ICurrentNonGUIApplication (or ICurrentApplication for graphic user interface applications), you can record and query the command line arguments of your application. Set the arguments by calling setArgs() with the arguments that were passed to the main function.

To query the number of arguments, use the member function ICurrentNonGUIApplication::argc(). This member function always returns a non-zero value because it has at least the name of the application as an argument.

To get the *n*th argument, use the member function ICurrentNonGUIApplication::argv(), where the argv(0) component is always the name of the application. Because argv() returns an IString, you can use all the functions provided by this class.

The following example demonstrates the use of command-line arguments, as well as the inclusion of header files into an application. The example accepts one argument, a string. It stores that string in an IText object, then outputs that string several times according to a macro defined in a header file. This example is intentionally convoluted to demonstrate the use of the different kinds of files listed in the previous table.

A VisualAge C++ application may use either a configuration file or a makefile. The following example will show you how to use both of these files to build your application.

The example consists of the following files:

- basic.cpp
- basic.hpp
- basic.h
- basic.icc or makefile

basic.cpp
The main() function creates a `MyClass` object called `currentData`. The `MyClass` constructor takes a reference of an ICurrentNonGUIApplication object as its argument. When the main() function creates the `MyClass` object, it saves the command line arguments with the setArgs() function.

The `MyClass` constructor accesses the name of the application by calling argv(0). The constructor accesses the value of the first argument by calling argv(1):

```
// IText-based Hello World!
#include "basic.hpp"
int main(int argc, char *argv[])
{
  // Create a MyClass object using the current application object
  // and save the command line arguments with the setArgs() function
  MyClass currentData(ICurrentNonGUIApplication::current().setArgs(argc, argv));
  // Output the name of the application (the first argument)
  cout << "Name of application: " << currentData.applicationName << endl;
  // Output the second argument
  for (int i = 1; i <= REPEAT; i++)
  {
     cout << i << ": " << currentData.stringArgument << endl;
  }
}
MyClass::MyClass(ICurrentNonGUIApplication& myApplication)
{
   // Store the first argument as the name of the application
   applicationName = myApplication.argv(0);
   // Store the second argument as the string to output.
   // If the second argument does not exist, then store
   //   a default string.
   if (myApplication.argc() > 1)
   {
     stringArgument = myApplication.argv(1);
   }
   else
   {
     stringArgument = "Default string";
   }
}
```

basic.hpp
This file contains the declaration of the `MyClass` class.

```
// basic.hpp
#include <iapp.hpp>
#include <itext.hpp>
#include <iostream.h>
#include "basic.h"
class MyClass : public IText
{
   public:
      MyClass(ICurrentNonGUIApplication& myApplication);
      IText applicationName;
      IText stringArgument;
};
```

basic.h
This file defines the macro used in this application.

```
// basic.h
#define REPEAT 7
```

basic.icc
Every application that uses the IBM Open Class must use the following options in its configuration file:

Option	Description
gen(rtti, yes)	Controls what run-time type information (RTTI) is generated. In this case, this option generates code that supports both the typeid and dynamic_cast operators.
WIN OS/2 link(linkWithMultithreadLib, yes)	Links the module being built to the multithread version of the VisualAge C++ run-time library.
AIX defaults(xlC_r)	Specifies the predefined option group xlC_r to use during compilation. The xlC_r string means the following: • Invokes the compiler so that source files are compiled as C++ language source code. • Sets macro names with the following: – define("_THREAD_SAFE", 1) – define("_AIX32_THREADS",1) – define("_AES_SOURCE",1) • Add the libraries libpthreads.a, libc_r.a, and /usr/lib/libc.a.

This configuration file uses several other optional options:

Option	Description
link(linkwithsharedlib, yes)	Links the module being built with the shared version of the VisualAge C++ run-time library.
WIN OS/2 link(extdictionary, no)	Does not search the extended dictionaries of libraries when the linker resolves external references. The *extended dictionary* is a list of module relationships within a library.
incl(searchpath, ".")	Searches the current directory to satisfy #include directives in the C and C++ source files.
macros(global)	Specifies that the source contains macros that apply to the whole project.

This configuration file uses three kinds of directives:

Directive	Description
option	Defines options for one of the tasks involved in a build.

Directive	Description
target	Specifies the name of the file that the build will produce. For example, this configuration file will produce a file called basic.exe.
source	Specifies files that are used as inputs to a build.

This example uses two classes from the IBM Open Class Library: IText and ICurrentNonGUIApplication. To use these classes in your application, you need to include the header files itext.hpp and iapp.hpp respectively. Instead of using an #include directive in the code, this example lists these header files in the configuration file.

This configuration file lists all the source files as well as the IBM Open Class header files in the scope of the target directive. Unlike a makefile, you do not specify any C++ file-dependency or processing information in a configuration file. In addition, you do not have to list these files in your source with the #include directive.

AIX

The following is the configuration file for AIX:

```
defaults(xlC_r),
link(linkwithsharedlib, yes),
gen(rtti, yes),
{
  target "basic"
  {
    option incl(searchpath,".")
    {
      option macros(global)
      {
        source type(hpp) "itext.hpp"
        source type(hpp) "iapp.hpp"
        source type(hpp) "basic.hpp"
        source type(h)   "basic.h"
      }
      source type(cpp) "basic.cpp"
    }
    source "libioc.a"
  }
}
```

WIN

OS/2 The following is the configuration file for Windows and OS/2:

```
option
  link(linkwithmultithreadlib, yes),
  link(linkwithsharedlib, yes),
  link(extdictionary, no),
  gen(rtti, yes),
  link(padding, no)
{
  target "basic.exe"
  {
    option incl(searchpath,".")
    {
      option macros(global)
```

```
      {
        source type(hpp) "itext.hpp"
        source type(hpp) "iapp.hpp"
        source type(hpp) "basic.hpp"
        source type(h)   "basic.h"
      }
      source type(cpp) "basic.cpp"
    }
  }
}
```

makefile

AIX Use the **make** command to compile this example with the following makefile. You do not need to put the names of any IBM Open Class header files in the make file:

```
# makefile for basic.exe
# -- Body --
all:      basic
basic:    basic.o
          xlC_r -o basic basic.o -lioc
basic.o:  basic.cpp basic.h
          xlC_r -c -qarch=com -qnotempinc -qnoinfo -qrtti=all \
          -I. -obasic.o -+ basic.cpp
clean:
          rm -f basic.o basic
```

WIN

OS/2 Use the **nmake** command to compile this example with the following makefile:

```
# makefile for basic.exe
#
# Make file assumptions:
#    - Environment variable INCLUDE contains paths to:
#       IBM Compiler target_directory\include;
#    - Environment variable LIB contains paths to:
#       IBM Compiler target_directory\lib;
#    - current directory will be used to store:
#        object, executable, and resource files
# -- Tool defintions --
ERASE=ERASE
GCPPC=ICC
GLINK=ICC
# -- Tool flags --
ICLCPPOPTS=/Gm+ /Gd+ /Gh+ /Ti+ /Fb+ /Q+
GCPPFLAGS=$(LOCALOPTS) $(ICLCPPOPTS)
GCPPLFLAGS=/Tdp /B"/debug /browse"
# -- Body --
all:  basic.exe
basic.exe:  basic.obj
      $(GLINK) $(GCPPLFLAGS) $(GCPPFLAGS) /Fe"basic.exe" basic.obj
basic.obj:  basic.cpp basic.hpp basic.h
      $(GCPPC) /C+ $(GCPPFLAGS) basic.cpp
# -- Clean --
clean:
        -$(ERASE) basic.exe
        -$(ERASE) basic.obj
```

Output

Assume that you execute this example with the following on the command line:

```
basic "Hello World!"
```

This will be the output:

```
Name of application: basic
1: Hello World!
2: Hello World!
3: Hello World!
4: Hello World!
5: Hello World!
6: Hello World!
7: Hello World!
```

RELATED CONCEPTS

IBM Open Class Overview
"User Interface Applications" on page 12
Configuration Files

RELATED TASKS

"Design an IBM Open Class Application"
"Design a User Interface Application" on page 15

Design an IBM Open Class Application

This section gives recommendations for designing Open Class Library applications. These general recommendations should not substitute for detailed design guidelines. Many of the topics listed here require a great deal of consideration when you design complex object-oriented applications. See the topic "Design a User Interface Application" on page 15 for more specific information on graphic user interface applications.

The Hello World sample application shipped with this product uses these design recommendations.

Including the IBM Open Class Library
To use the classes, functions and operators available in the IBM Open Class Library, you must include the parts of the library's interface that you need in your C++ source code. To include an interface, you must either:

- Specify the header file in a source configuration file directive that applies to any build target using this interface.
- Include the following statement in any file using this interface:

  ```
  #include <filename>
  ```

  where `filename` is the name of the header file.

Then, in the body of your program, you can use a class, function or operator defined in the header file, as well as derive new classes and overload the functions and operators. See the topic Libraries, Headers, and Conventions for more information about how the IBM Open Class names its header files and classes.

Create Your Own Classes
Most applications require you to derive new classes from existing classes. You derive new classes to inherit implementation details from a base class.

Do not derive from a particular base class unless you have a good reason to do so. When creating data type or settings classes, do not derive from a base class, unless for some reason your new class must derive from one of your own base classes.

Choose Multiple Inheritance or Composition
It is easier to inherit from multiple classes when you design simple applications. Because all of the functions from the derived classes are immediately available, you can easily use them as-is and not override them.

However, as your application evolves into a more complex application, it can be difficult to anticipate how changes in the functions of the inherited classes will affect the derived class.

Generally, if the class you design has an "is-a" relationship with, for example, a frame window, then it should inherit from the IFrameWindow class. Inheriting from IFrameWindow is typical, but not necessary in most cases. For example, a "has-a" relationship is often fine with frame windows. However, if the class has a "has-a" relationship, for example, command handler, the derived frame window class should contain an ICommandConnectionTo<> template object as a data member, or an object derived from ICommandHandler. It should not inherit from the command handler class. The Hello World version 3 sample application provides an example.

Override Virtual Functions
When you override inherited member functions, such as the ICommandHandler::command() function, that are defined as virtual, declare the overriding function as virtual too. This improves the readability of the inheriting class by saving the reader from having to search up the inheritance chain to discover that the function was originally defined as virtual.

Delete Objects Created with New
If you create objects dynamically by using the new operator, delete them by using the delete operator. If an object is composed of dynamically created objects, that is, you create the composed objects with the new operator in the constructor of the composing object, then you should delete the object in the destructor of the composing object.

The following are exceptions to this rule:

- Classes that use the autoDelete behavior of IWindow derived classes. See IWindow::setAutoDeleteObject().
- IContainerColumn and IContainerObject. You can have the IBM Open Class delete these for you by calling IContainerControl::setDeleteColumnsOnClose() and IContainerControl::setDeleteObjectsOnClose().
- Classes derived from IMRefCounted.
- ICountedPointerTo<> template objects.
- Objects that you pass to functions with "adopt" semantics, such as IThreadFn* pointers you pass to IThread, and IGrafBundle* pointers you pass to INonGUIThread, IThread, and IMGraphic::adoptBundle().

Define Strings Outside the Executable Program
The values of strings in applications vary by user because of preference or national language. Therefore, you should define strings outside the application.

To use strings outside of a non-GUI application, use a message file.

RELATED CONCEPTS

IBM Open Class Overview
"IBM Open Class Applications" on page 4

RELATED TASKS

"Design a User Interface Application" on page 15

RELATED REFERENCES

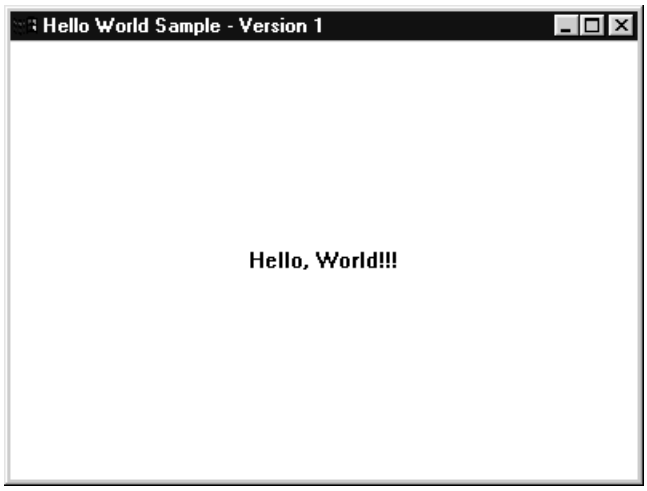
IBM Open Class Libraries, Headers and Conventions

User Interface Applications

An easy way to understand how the classes and objects work together in a user interface application is to look at a simple application called Hello World Version 1. This application has two basic user interface components:

- A standard frame window with a title bar, system menu button, border, and minimize and maximize buttons.
- The rest of the window, called the client area, containing the phrase "Hello World!!!"

The main window for Hello World Version 1 looks like this:



One source file, the .cpp file, is required for this application.

Creating a C++ Source File

The Hello World samples, versions 1 through 6, illustrates many Open Class Library features. Hello World version 1 has only a .cpp file. This file is the C++ source file used by VisualAge C++ to generate the executable part of this application. A copy of the "Hello World" version 1 application is included in the online samples.

The listing of the C++ source file for the Hello World version 1 application follows:

```
#include <iapp.hpp>
#include <istattxt.hpp>
#include <iframe.hpp>
#include "ahellow1.h"
/************************************************************
* main                                                      *
*  - Application entry point for Hello World Version 1.     *
*                                                           *
* Creates a new object mainWindow of class IFrameWindow     *
* Creates a new object hello of class IStaticText           *
* Sets the static text value and aligns it                  *
```

```
* Sets the static text as the client of the mainWindow      *
* Sets the size of mainWindow                               *
* Sets the window focus to mainWindow                       *
* Displays the mainWindow                                   *
* Starts the events processing for the application          *
************************************************************/
int main()
{
  IFrameWindow mainWindow (WND_MAIN);
  IStaticText hello(WND_HELLO, &mainWindow, &mainWindow);
  hello.setText(STR_HELLO);
  hello.setAlignment(IStaticText::centerCenter);
  mainWindow.setClient(&hello);
  mainWindow.sizeTo(ISize(400,300));
  mainWindow.setFocus();
  mainWindow.show();
  IApplication::current().run();
  return 0;
} /* end main() */
```

This application creates the following objects:

Object	Description
mainWindow	This IFrameWindow object is the main window for the application.
hello	This is the static text control (IStaticText) object that displays the string “Hello World!!!” which was loaded from the .rc file
ISize object	An unnamed temporary ISize object used in the call to mainWindow.sizeTo()
ICurrentApplication object	The call to IApplication::current() returns a reference to the current application, which is an object of the class ICurrentApplication

In the main function you create the primary application window, call its functions to change settings, such as size or position, call its inherited frame window functions to give it focus and have it displayed, and call ICurrentApplication::run() to begin event processing.

To create a user-interface application with more functionality than the Hello World version 1 sample, you generally create a class that represents your main window. Usually this class derives from IFrameWindow. The IFrameWindow class, as well as the IWindow class, have many functions you can call that perform general tasks like resizing the border of your main window.

In the constructor of your main window class you can create *controls* (such as push buttons or entry fields) and *event handlers* (functions that process events like the selection of a push button).
The Open Class Library provides you with virtual functions that you can override. For example, the ICommandHandler class (a class that processes application command and system command events) has a virtual function called command(). You can override this function to process events such as the selection of a command from a menu bar.

You can create objects of other classes based on user input.

In the destructor of your main window class, you stop your event handlers and delete any composed objects that you created using operator new, except those IWindow objects that the Open Class Library will delete for you. Specify these objects by calling IWindow::setAutoDeleteObject().

Start Event Processing
Starting *event processing* allows the events generated by the system, user, or application to be processed by event handlers or by default behavior provided by the system.

To start event processing for a C++ application using the Open Class Library, follow these steps:

1. Obtain a reference to ICurrentApplication by using the static member function IApplication::current(). Objects of the ICurrentApplication class represent the application that is currently running.
2. Call ICurrentApplication::run() to execute your application.

The following example comes from the Hello World version 1 source file:

```
IApplication::current().run();
```

By default, the call to ICurrentApplication::run() does not return until all primary windows have been closed. Without this call, the application window you created displays only briefly before the main function exits, thereby ending the application prematurely.

Load Resources into an Application
Resources are user-interface components, such as text strings, icons, and bitmaps. (These resources are not the window-related resources used by X and Motif, such as the ones you can place in your .Xdefaults file.) The following are the basic steps to load resources into your application:

1. Decide what resources to separate from the code. Many can be hardcoded into the program, such as strings and accelerator keys. Others have to be in a resource file such as bitmaps and icons (unless you decide to load the images instead directly from the bitmap and icon files), and help tables.
2. Code those resources in a .rc file.
3. Compile the resource file using the resource compiler.
4. Link the resulting binary resource file to either the executable file or to a shared library.
5. Add code to your application to use the resources you created.

See the concept Resources for more information.

The Open Class Library loads resources where necessary for an application. Use the ICurrentApplication member function setUserResourceLibrary() to identify which resource library will be used if none is specified on a call that loads a resource. The following example uses the name passed as a program parameter as the name of the application's default resource library.

```
// Main function with arguments
int main(int argc, char *argv[])
{
  // Save the command line arguments
  // in the current application object
  IApplication::current().
    setArgs(argc, argv);
  IString dllname(IApplication::current().argv(1));
  // Get current application
```

```
  // and set the name of resource DLL
  IApplication::current().
    setUserResourceLibrary(dllname);
  // Create main window
  AHelloWindow mainWindow (WND_MAIN);
  // Start event processing
  // for the application
  IApplication::current().run();
  return 0;
} /* end main */
```

The Open Class Library tries to create a frame window by loading a dialog with the `WND_MAIN` ID from the default user resource library, specified on the call to ICurrentApplication::setUserResourceLibrary().

AIX

This construction of a frame window does not apply on AIX since this platform does not support dialogs defined with resource libraries.

You can also determine the current user resource library by calling the userResourceLibrary() member function. The following code displays the name of the user resource library in the client window of the frame window:

```
// Call IFrameWindow constructor
AHelloWindow :: AHelloWindow(unsigned long windowId)
  : IFrameWindow(windowId)
{
  // Create static text control
  // Pass in myself as owner & parent
  hello=new IStaticText(WND_HELLO,
         this, this);
  // Set text in static text control to library name
  hello->setText(
    IApplication::current().
    userResourceLibrary().fileName());
  // Set alignment to center in both directions
  hello->setAlignment(
     IStaticText::centerCenter);
  // Set hello control as client window
  setClient(hello);
  // Set focus to main window
  setFocus();
  // Set to show main window
  show();
} /* end AHelloWindow :: AHelloWindow(...) */
```

RELATED CONCEPTS

IBM Open Class Overview
Resources

RELATED TASKS

"Design an IBM Open Class Application" on page 10
"Design a User Interface Application"

Design a User Interface Application

This section gives recommendations for designing user interface applications using the IBM Open Class. See the topic "IBM Open Class Applications" on page 4 for more general information.

Create Your Own Classes
Most applications require you to derive new classes from existing classes. You derive new classes to inherit implementation details from a base class.

The following table provides a starting point to determine which base class to use:

Class Functionality	Derived From
Attribute	IAttribute (for window attributes)
Canvas class	ICanvas
Control	IWindow or ITextControl
Data type	-
Event	IEvent
Primary or secondary window	IFrameWindow
Settings	-
Style	IBitFlag
Window behavior	IHandler or one of its derived classes

Do not derive from a particular base class unless you have a good reason to do so. When creating data type or settings classes, do not derive from a base class, unless for some reason your new class must derive from one of your own base classes.

For window attributes, derive from IAttribute, since the IWindow interface supports the addition and removal of IAttribute objects. For other named attributes, IAttribute is also a handle. For arbitrary, unnamed attributes attached to your own classes, you do not need to derive from an Open Class Library base class.

Define Strings Outside the Executable Program
The values of strings in applications vary by user because of preference or national language. Therefore, you should define strings outside the application.

In Windows and OS/2 this capability is provided by using Windows or OS/2 resource compiler (.rc) files. This format lets you use descriptive tags to identify tables of strings and associate them with unique IDs in your application. In AIX, the default format used by the resource compiler is the OS/2 format.

The syntax for the resource files on the various platforms are very similar, but the keywords are different. Also, this product's resource compiler handles extended menu syntax and keywords. Refer to the Resource Compiler - Syntax topic in the Tools and Utilities section of the reference for information on the resource file syntax. Refer to the Resource Conversion Utility - Overview topic in the Tools and Utilities section of the reference for information on converting your resource files.

Hello World versions 1 and 2 show how to use Open Class Library functions to reference strings from resource files.

Define Menus in Resource Files
Use resource files to define menu bars, pop-up menus, and accelerators. By using the same ID for the menu bar and the frame window, you can define the layout, menu item text, and accelerator key definitions external to application logic. Hello World version 3 and version 4 demonstrate this feature.

Use Canvases Instead of Dialog Templates
Windows and OS/2 provide support for dialog templates, which are "layouts" of frame windows and controls. This support is not available in Motif and, therefore, is not portable. Instead, use the canvas classes, such as IMultiCellCanvas and ISetCanvas, for designing portable dialogs across Windows, Motif, and OS/2. Hello World version 4 demonstrates this feature. Note that there are also other features of canvases, such as automatic layout, that compensate for internationalization text changes and font differences between platforms.

Defining the Client Window ID
To specify the client window in a way that is portable, create the window with IC_FRAME_CLIENT_ID (defined in icconst.h) as its identifier, and call the IFrameWindow::setClient() function.

RELATED CONCEPTS

IBM Open Class Overview
"IBM Open Class Applications" on page 4
"User Interface Applications" on page 12

RELATED TASKS

"Design an IBM Open Class Application" on page 10
"Create Cross-Platform Applications"
"Compile and Build Open Class Applications" on page 22

RELATED REFERENCES

Resource Compiler - Syntax
Resource Conversion Utility - Overview

Create Cross-Platform Applications

There are many things to consider when you create applications that are portable across OS/2, Windows, and AIX. This section discusses how the Open Class Library differs between platforms and how you can include nonportable functions in a portable application. For information on converting resource and bitmap files between OS/2, Windows, and AIX, see the topic Convert Application Resources.

Open Class Library Differences between Platforms
The Open Class Library for AIX does not support the following items which are supported in the Open Class Library for OS/2 and Windows:

- Dialog template (DLGTEMPLATE) resources and the ability to construct an IFrameWindow from a dialog template resource. (If you port OS/2 or Windows programs that use dialog templates, you must convert them to use the canvas classes.)
- All Dynamic Data Exchange (IDDE*) classes. There is no support for Dynamic Data Exchange under Motif.
- IGroupBox does not support a transparent background. This may cause painting conflicts with sibling windows. For portable applications, use ICanvas or a class derived from ICanvas, which has methods that let you draw a box around the canvas and label your canvas.
- IWindow::setParent().
- ITabControl.

- IContainerControl is limited.
- Multimedia classes and depreciated 2D graphics classes.

Platform Specific and Ignored Classes and Member Functions
Some classes and member functions from the Open Class Library are platform specific. Attempts to use these classes and member functions for platforms other than those platforms they were designed for will result in a compile-time error.

The Open Class Library makes these classes and member functions platform specific by using the #ifdef and #endif preprocessor directives, and the macro symbols listed in the table at the bottom of this page.

Also, there are some classes and member functions that are ignored on certain platforms. This means that you will get valid compile results because the class or member function is ignored. An example is using the IRadioButton::disableAutoSelect() function in AIX. The autoselect style is always in effect for Motif. Therefore, when your program calls disableAutoSelect(), the Open Class Library does nothing. Many of these ignored classes and member functions result from "look and feel" differences between OS/2 Presentation Manager, Windows, and Motif.

The Open Class Library notes which classes and functions are ignored with flags in the .hpp files. To see if your application contains any of these classes and member functions, include the option define(IC_MOTIF_FLAGNOP) directive in your application's configuration file. See the topic "Obsolete or Ignored Member Functions" on page 31 for more information.

This causes all of the ignored classes and member functions to not be declared, so any occurrence of an ignored class or member function in your application produces a compiler error. Note that you cannot link using object files built with IC_MOTIF_FLAGNOP or other options that notes which classes and member functions are ignored. The use of these compile-time checks causes the compiler to generate code that does not match the IOC runtime libraries.

Include Nonportable Functions in Portable Applications
You may want to take advantage of system-specific APIs while developing an application with Open Class Library. For example, in Create Your Own Handler Sample, the X-Windows timer function gets the system time. However, these calls prevent the application from building on OS/2 or Windows. To overcome this, the application also provides the corresponding OS/2 and Windows function calls. The conditional use of these calls is accomplished by using the compiler preprocessor directives, #ifdef and #endif. In this case, you need the following symbols in the #ifdef statement: IC_MOTIF, IC_WIN, and IC_PM.

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

The Open Class header files environment symbols are defined in icomdefs.h. To get these symbols, include the icomdefs.h. file. The following lists the symbols and the corresponding platforms the symbol is defined when compiling:

Symbol	Platform
IC_PM	OS/2
IC_WIN	WIN

Symbol	Platform
IC_MOTIF	AIX
IC_PMWIN	OS/2 WIN
IC_MOTIFPM	AIX OS/2
IC_MOTIFWIN	AIX WIN

RELATED CONCEPTS

"Obsolete or Ignored Member Functions" on page 31

RELATED TASKS

Convert Application Resources

Map Coordinate Systems Across Platforms

Unfortunately all operating systems do not position windows on the display using the same coordinate system. In the Windows and AIX operating systems, you position a window relative to the upper-left corner of the window it is contained in. Under OS/2, you position a window relative to the lower-left corner of its containing window.

Use ICoordinateSystem to Set Your Application's Orientation
To facilitate developing cross platform applications, the Open Class Library provides you with the ICoordinateSystem class. This class allows you to develop your application based on one of the two coordinate systems, and provides functions to assist you in porting it to a platform with a different orientation. The ICoordinateSystem::EOrientation enum has two values, kOriginUpperLeft and kOriginLowerLeft.

To develop a portable application, we recommend that you pick one of the two orientations and code all of your windows using this orientation.To change the way the Open Class Library interprets coordinates you must call the ICoordinateSystem function setApplicationOrientation function, passing the EOrientation you wish to use. This will cause the Open Class Library to interpret all input parameters, and return objects based on this orientation. For example, to code all coordinates based on the lower-left corner use the following line of code:

```
ICoordinateSystem::setApplicationOrientation(ICoordinateSystem::kOriginLowerLeft);
```

You must make this call prior to calling any Open Class Library function that uses the window coordinates.

If an application does not call ICoordinateSystem::setApplicationOrientation(), then the orientation defaults to upper-left.

Calling this function after IWindow objects have been created may cause unpredictable results. If your design calls for using an orientation other than the Open Class Library default of native orientation, you should set the orientation early in your program.

This function does not affect the processing of dialog templates loaded from resource files. The coordinates in resource files are always interpreted in native system coordinates when the dialog is being loaded.

You can construct an IExtendedRootGrafPort object with an ICoordinateSystem::EOrientation enumerated type for drawing 2D graphics with the coordinate system of your choice. If you do not specify an enumerator value when constructing the IExtendedRootGrafPort object, it will use the application orientation by default. You can set the orientation by calling ICoordinateSystem::setApplicationOrientation().

Use Platform Dependent Coordinates
There may be situations where you will need to deal with the native coordinate system. For example, if you develop your own control and make the calls to the operating system to create the control or to move and size it, you must specify the window's position and size in native coordinates. The Open Class Library also contains functions to help you with these conversions.

If you are extending the library by writing classes for custom controls or providing other reusable classes based on Open Class Library, then you should consider using this class to provide coordinate mapping for your users. Coordinate conversion is needed when you make calls to system routines which accept points or rectangles. It also may be needed if you rely on a specific orientation to perform layout of windows or graphics.

You can determine the native and application orientations by calling the ICoordinateSystem functions nativeOrientation() and applicationOrientation() respectively. When they return different values you need to convert the coordinates of objects that are passed to and returned by you. The ICoordinateSystem function isConversionNeeded() is the best way to determine if you must make conversions. It returns true if the native and application orientations are different.

When developing cross-platform applications, you may need to use the following member functions to manipulate the orientation of your application objects:

- Use ICoordinateSystem::convertToApplication() to convert a point or rectangle from the native coordinate orientation to the application orientation.

  This function returns a point or rectangle in application coordinates. This is computed from a point or rectangle expressed in native orientation and a reference size (the size of the coordinate space in which the native point coordinate is expressed).
- Use ICoordinateSystem::convertToNative() to convert a point or rectangle from the application coordinate orientation to the native orientation. This function returns a point or rectangle in native coordinates.
- Use IWindow::nativeRect() to get a IRectangle object that represents the position and size of a window in the native coordinates. (IWindow::rect() returns the position and size of the window in reference to the application's coordinate system.)

Use Portable Rectangles
Various graphics and windowing classes, as well as their member functions, use rectangles so when porting applications, you need to understand the different coordinate system mappings.

Objects of the IRectangle class represent a rectangular area defined by two points that form opposite corners of the rectangle. These two points are referred to as the minimum and maximum points. IRectangle objects are designed to be usable independently of the coordinate system in use. The minimum, or origin, is defined as the point with the lowest coordinate values. Therefore, in a coordinate space where (0, 0) is the upper left and increasing a point's coordinate value moves it to

the right and down, the minimum point of an IRectangle is the top-left corner and the maximum corner is the lower-right corner. Conversely, in a coordinate space where (0, 0) is the lower-left corner and increasing a point's coordinate value moves it to the right and up, the minimum corner of an IRectangle is the lower left, and the maximum corner is the top right.

The original functions provided in IRectangle to return points and coordinates had names like lowerLeft() and upperRight(). So, to get the origin of a rectangle on OS/2 you would call lowerLeft(). Unfortunately this is not the origin on Windows or AIX.

IRectangle is a data type class that is based on a mathematical coordinate system (where *y* values increment from bottom to top). This coordinate system also corresponds to the one used by OS/2, but IRectangle does not consider the application orientation which deals with window coordinates only. As a result, the implementation of IRectangle::lowerLeft() returns the point with the smallest X and Y values in the rectangle. If you use the IRectangle functions in the context of a Windows coordinate system, IRectangle::lowerLeft() returns what would be the upper left corner of the rectangle. So IRectangle::lowerLeft() does return the origin of a windowing rectangle on all platforms, but it is extremely confusing when not using a lower-left coordinate system.

IRectangle now has new functions that allow you to get information in an orientation aware manner. For instance there are now functions named minXMinY(), minXCenterY(), and minXMaxY(). To write portable applications you should use these new functions exclusively.

Set Window Positions
You use the IWindow member functions for window positioning to set and query the size and position of windows. Unless otherwise noted, the orientation of the coordinates accepted and returned by these members is the application orientation.

IWindow::mapPoint() for ITitle::handle() mapping to the desktop returns the wrong data. This appears to return native coordinates relative to the application when the default is the OS/2 coordinate system. These problems arise since on Windows, the following is true:

```
frame->handle() == title->handle();
```

RELATED CONCEPTS

2D Coordinate System and Data Types

RELATED TASKS

"Create Cross-Platform Applications" on page 17

Compile and Build Open Class Applications

Compile Options
All Open Class executables must use the following options in their configuration files or makefiles:

Configuration File Option	Makefile Option	Description
gen(rtti, yes)	-qrtti=all	Controls what run-time type information (RTTI) is generated. In this case, this option generates code that supports both the typeid and dynamic_cast operators.
WIN OS/2 link(linkWithMultithreadLib, yes)	-lioc	Links the module being built to the multithread version of the VisualAge C++ run-time library.
AIX defaults(xlC_r)	xlC_r	Specifies the predefined option group xlC_r to use during compilation. The xlC_r string means the following: • Invokes the compiler so that source files are compiled as C++ language source code. • Sets macro names with the following: – define("_THREAD_SAFE", 1) – define("_AIX32_THREADS",1) – define("_AES_SOURCE",1) • Add the libraries libpthreads.a, libc_r.a, and /usr/lib/libc.a.

AIX

If an Open Class application or an application which uses POSIX threads is built with a non-optimized configuration file, an error similar to the following may be generated:

```
"/usr/include/pthread.h", line 552.17:(S)CPPPC0403:
"_mutex_global_np" is already defined.
```

This may occur because the system header file pthread.h contains the definition of a variable called _mutex_global_np. Some of the Open Class header files include pthread.h. This causes duplicate definitions when these headers are included in separate source files which are built into the same target. To resolve this, add the following to the configuration file as one of the sources for the appropriate target:

```
option macros(global,yes)
{
  source type(h) 'pthread.h'
}
```

AIX

Build with the Motif 1.2 Compatibility Library on AIX 4.3
The Open Class Libraries are built using Motif 1.2 and X11R5. On AIX 4.3 the operating system has made Motif 2.1 and X11R6 the default. An application that uses IBM Open Class (and thus Motif 1.2) cannot use Motif 2.1. These two levels of Motif are not binary compatible.

Thus for a user to mix native X-widgets and IBM Open Class widgets, on AIX 4.3 the user will have to compile and link to the *Motif 1.2 compatibility library*. To use these libraries read the information in the /usr/lpp/X11/readme file.

Avoid Reserved Pragma Priority Values
The Open Class Library reserves the use of #pragma priority values in the range of -2147482624 through -2147481600. The C++ compiler reserves the range below that. As a result, avoid using a #pragma priority value less than -2147481599 (this is equivalent to INT_MIN + 2048) to control the order of static object construction in your Open Class Library application.

Open Class Library Error and Exception Output
Although Open Class is designed to catch as many errors as possible during the compilation and link steps, some errors can only be detected at run time. The classes in Open Class throw C++ exceptions to indicate runtime errors. Errors messages describing the exception can be seen while debugging, or can be seen in trace output sent to STDOUT, STDERR or a queue; trace output is only seen if you have turned tracing on. Your own classes can also throw C++ exceptions and output trace information in the same way, by using the IException and ITrace classes.

Link an Application to the Open Class Library

WIN OS/2 To use dynamic linking, specify link(linkWithSharedLib, yes). To use static linking specify link(linkWithSharedLib, no). When statically linking on OS/2 and Windows, this option automatically pulls in the necessary Open Class Library static link libraries.

WIN

OS/2 You do not need to specify which libraries to use because this happens automatically via #pragma library statements. Only if you build with the link(defaultLibs,no) will you need to manually include the names of the import libraries (if dynamically linking) or static libraries (if statically linking).

AIX

On AIX, you must specify exactly which libraries should be used. You should link an IBM Open Class application against libioc.a.

The following additional rules apply when you build your application with the dynamic libraries, instead of the static object libraries:

1. A DLL using the Open Class Library must link dynamically to the Open Class Library code (that is, you must link with the Open Class Library import libraries). In other words, if you build a DLL that uses Open Class, then you must not build this DLL with the link(linkWithSharedLib,no) option.

2. An .exe using the Open Class Library and calling a DLL that also uses the class library must link dynamically to the Open Class Library (that is, you must link with the Open Class Library import libraries).
3. An .exe or .dll file should not link both dynamically and statically to the Open Class Library code.

AIX

Link your AIX application to the Open Class Library by specifying either the import library or the following static libraries at link time:

- Import library:
  - libioc.a
- Static libraries:
  - libiocns.a (application control, streams, test framework, and everything else not listed below)
  - libiocclns.a (collection classes)
  - libiocunis.a (user interface and 2D graphics)

RELATED CONCEPTS

IBM Open Class Overview
"IBM Open Class Applications" on page 4

RELATED TASKS

"Build a 64-Bit Enabled Application"

RELATED REFERENCES

IBM Open Class Libraries, Headers and Conventions

Build a 64-Bit Enabled Application

You can build in 64-bit mode IBM Open Class applications that do not contain any User Interface or 2D Graphics code.

New typedefs have been added for creating portable applications that can be built using either the 32-bit or 64-bit definitions of longs. When you build your application in 64-bit mode, the macro __IOC_64BIT is set within the IBM Open Class Library. This macro controls the use of the typedefs for long or integer variables. See an example of this in the istring.hpp file found in the include directory.

To build a 64-bit application, use either one of these build options:

- OBJECT_MODE environment variable (for example, export OBJECT_MODE=64)
- gen(objectMode,64) configuration option
- -q64 compiler option

RELATED CONCEPTS

IBM Open Class Overview
"IBM Open Class Applications" on page 4

RELATED TASKS

"Compile and Build Open Class Applications" on page 22

Package and Distribute an IBM Open Class Application

An important factor in determining how you package and distribute your application is whether you link statically or dynamically to VisualAge C++ code. Linking statically creates a larger executable program but minimizes the number of files you have to distribute. Linking dynamically is better if you build several programs since this allows a single copy of VisualAge C++ code to be shared instead of a copy linked into each executable file.

An application built with the IBM Open Class (IOC) needs the following files at runtime:

- IOC runtime code: See IBM Open Class Libraries, Headers, and Conventions for the file names of the runtime libraries.
- IOC language files
  - Resource library: The libvacocres.o (AIX) and the cpporr40.dll (Windows and OS/2) files store resources that the library provides, such as bitmaps and text for standard toolbar buttons, bitmaps for animated buttons, container icons, direct manipulation icons, and system pointers.
  - Message file or catalog: The ibmvaccl.cat (AIX) and the cppaoi40.msg (Windows and OS/2) supply text, primarily for the IBM Open Class exceptions. If these messages are not available at runtime, the IOC defaults to using English strings for the text it needs.

See Package and Deliver Your Finished Application for additional VisualAge C++ files you may need. These include the following:

- C/C++ runtime code
- AIX WIN IPF runtime code for help support

Package a Dynamically Linked Application

AIX When packaging your application you must include the VisualAge C++ fileset(s) that contains the IBM Open Class runtime libraries, and optionally the filesets containing the language files libvacocres.o and ibmvaccl.cat. The IBM Open Class libraries cannot be renamed or individually redistributed. For example, you do not distribute libvacbase5.a, but rather the fileset that this library belongs to.

WIN

OS/2 You must rename the runtime .dll files and the resource library cpporr40.dll in order to ship them. Use the **dllrname** tool shipped with VisualAge C++ to rename these files.

WIN

The first four characters of cppoqi40.dll (the library that is needed primarily by the PM-compatible versions of the following user interface classes: IContainerControl, IProgressIndicator, ISlider, INumericSpinButton, ITextSpinButton, and INotebook) must be the same as the first four characters of the renamed user interface classes library (cppoui40.dll).

WIN

OS/2 In addition to renaming the Open Class resource library cpporr40.dll, add a call to ICurrentApplication::setResourceLibrary to the beginning of your program. Specify the new resource file library name as an argument to the call. You can also

set the environment variable ICLUI_RESLIB to change the default name of this DLL. The function call is better for production-level applications.

WIN

OS/2 You must rename the the message file cppaoi40.msg in order to ship it. To identify the new name, add a call to IMessageText::setMessageFile to the beginning of your program. Specify the new name as an argument to the call.

Package a Statically Linked Application

WIN You cannot statically link to the PM-compatible code in cppoqi40.dll. The user interface classes dynamically load this DLL when needed. If your program needs this DLL to run you must ship it. In this case rename the file using **dllrname** so its first four characters match the first four characters of the file name of your executable program. The last four characters of the DLL must remain "qi40."

You can either ship the IOC resource library as described in the previous section (for AIX ship the fileset containing libvacocres.o, for Windows or OS/2, rename cpporr40.dll), or add the resources to your own resource library.

Follow these steps to include IBM Open Class resources into your application's resource library:

1. Add all the resources that your application requires, including those built into libvacocres.o (AIX) or cpporr40.dll (Windows and OS/2), into your application resource file. IBM Open Class resources are located in the following directories:
   - WIN OS/2 source/res
   - AIX /usr/vacpp/iocsrc/res/aix

   To ensure that you have not omitted any resource, we recommend that you include all resources that the IBM Open Class provides.
2. Ensure that the IDs of your resources do not duplicate those of the IBM Open Class resources you are adding to your resource file.
3. Add a call to ICurrentApplication::setResourceLibrary to the beginning of your program. Specify the name of the resource library as an argument to the call. (Use a value of 0 to specify the application executable file.)

WIN

OS/2 You must rename the the message file cppaoi40.msg in order to ship it. To identify the new name, add a call to IMessageText::setMessageFile to the beginning of your program. Specify the new name as an argument to the call.

AIX

You should ship the fileset containing the IOC message catalog, ibmvaccl.cat.

RELATED TASKS

"Compile and Build Open Class Applications" on page 22
Package and Deliver Your Finished Application

RELATED REFERENCES

IBM Open Class Libraries, Headers, and Conventions

Build the IBM Open Class Library Source Code for Debugging Purposes

You can build debuggable versions of the IBM Open Class (IOC) libraries.

Conditions
The IBM Open Class source is provided only to help you debug your code. You cannot redistribute the source or binaries you build from these source files.

Prerequisites
You can build the IOC libraries either with the incremental or the batch compiler.

- AIX To build with the batch compiler you will need **xlC_r** and GNU Make 3.77 (or above) (ftp://ftp.gnu.org/pub/gnu/make)
  Note: GNU Make is not officially supported by IBM.
- To build with the incremental compiler you will need **vacide**.

Operating System Prerequisites
In order to build the IBM Open Class libraries, you must have a minimum amount of disk space and memory available as listed in the following table:

Component	Requirements
Disk space	~600MB
Memory	~512MB

Environment Variable Settings
The IOC build procedure uses the following environment variables:

Environment Variable	Description	Default Value When Undefined
OPENCLASSBUILDTOS	Defines the build target operating system. The following are the possible values: • aix42 • aix43	You must define this variable
OPENCLASSROOT	If this is defined, the IOC source files are assumed to be stored in the directory $(OPENCLASSROOT)/iocsrc.	If this is not defined, the IOC source files are assumed to be stored in the directory /usr/vacpp/iocsrc.
OPENCLASSBUILDROOT	Defines where the newly built libraries are stored. New libraries are stored in the following directory: $(OPENCLASSBUILDROOT)/lib/$(OPENCLASSBUILDTOS) Files generated during the build are stored in the following directory: $(OPENCLASSBUILDROOT)/$(OPENCLASSBUILDTOS).bin	The current working directory

Environment Variable	Description	Default Value When Undefined
OBJECT_MODE	Specifies whether you are building 32-bit or 64-bit objects.	32-bit

IOC Source Directory Hierarchy and Installation Targets

In the standard installation of VisualAge C++, you can find the IBM Open Class source tree in the /usr/vacpp/iocsrc directory. In the following tables that describe the directory hierarchy and installation targets, the top level IOC source directory is $(top_srcdir).

Non-Graphical Libraries

Library	Description	Directory	Target	Library File Name
Core/Base	Services such as multithreading, resource locking, object streaming, internationalization support, and file system access.	$(top_srcdir)/core	core	libvacbase5.a
Collection	An alternative collection library to the ANSI Standard Template Library. It supports notification and streaming.	$(top_srcdir)/collect	collect	libvaccl5.a
Test Framework	A framework for creating and running test cases.	$(top_srcdir)/testfw	testfw	libvactestfw5.a
File I/O Stream	I/O stream support for files.	$(top_srcdir)/fstream	fstream	libvacfstrm5.a

Graphical Libraries

Library	Description	Directory	Target	Library File Name
2D Graphics	Creates and manipulates 2D graphics.	$(top_srcdir)/graph2d	graph2d	libvacgraph2d5.a
User Interface	A framework for creating graphical user interfaces.	$(top_srcdir)/ui	ui	libvacui5.a

Import and Static Libraries

The following libraries do not have any targets; they are generated automatically by invoking any of the above targets:

Library	Library File Name
Import Library	libioc.a
Static non-GUI Library	libiocns.a
Static GUI Library	libiocuins.a

Library	Library File Name
Static Collection Library	libiocclns.a

Build with the Batch Compiler
To build a debuggable IOC library with the batch compiler, call **make** with the master makefile, called Makefile. For example, the following command creates the Core library in the output tree:

```
make -f /usr/vacpp/iocsrc/Makefile core
```

If you do not supply a target, the default behavior is to rebuild the entire IOC library.

Build with the Incremental Compiler
If you want to use the incremental compiler, use the **vacide** command. To open the Open Class Library project, invoke the following command:

```
vacide /usr/vacpp/iocsrc/IOC.icp
```

Within the IDE, choose the library you want to build in the Overview page.

Example of Building an IOC Library
For example, if you want to create a debuggable IOC library in the /home/userid/iocout directory, invoke the following commands:

- `slibclean` (in case an application is still using the libraries)
- `export OPENCLASSBUILDTOS=aix42`
- `export OPENCLASSBUILDROOT=/home/userid/iocout`
- `make -f /usr/vacpp/iocsrc/Makefile`
  or
  `vacide /usr/vacpp/iocsrc/IOC.icp`

After the build is completed, the object files will be in the directory /home/userid/iocout/aix42.bin, and the libraries will be in the directory /home/userid/iocout/lib/aix42.

To use the new libraries for debugging, set the LIBPATH environment variable to include the directory /home/userid/iocout/lib/aix42:

```
export LIBPATH=/home/userid/iocout/lib/aix42:$LIBPATH
```

Build Messages
When building the IOC libraries, you might encounter some Duplicated Symbol warnings. These warnings are normal and can be safely ignored.

AIX4.3 Considerations
AIX4.3 supports two object modes: 32-bit and 64-bit. You can build only the non-graphical IOC libraries using either object mode. To compile the IOC non-graphical libraries using the 64-bit object mode, set the OBJECT_MODE environment variable to 64 (for example: export OBJECT_MODE=64).

AIX4.3 also supports two versions of Motif: 1.2 and 2.1. However, the IOC graphical libraries support only Motif 1.2. The IOC graphical libraries require the X11.compat.adt.Motif12 fileset to get the proper headers and import libraries.

RELATED CONCEPTS

IBM Open Class Overview

RELATED TASKS

Compile and Build an IBM Open Class Application

Work with the IBM Open Class Samples

VisualAge C++ ships samples that demonstrate the use of the IBM Open Class. You can use the samples to learn the IBM Open Class by example. The samples can show you how to properly use a class within the context of an application. You can also use the samples as a starting-point for an application you want to develop. Find a sample that closely resembles your desired application, and add or revise code.

Copy Samples

AIX To compile any sample, you must create a copy in a directory that you have access to. For instance, enter the following commands in a command shell to copy the IBM Open Class sample called "notebook":

- Go to your userid's top directory:
  ```
  cd ~
  ```
- Create a directory to store the sample:
  ```
  mkdir -p samples/ioc/notebook
  cd samples/ioc/notebook
  ```
- Recursively copy all the files (including sub-directories) to the destination directory:
  ```
  cp -r /usr/vacpp/samples/ioc/notebook/* .
  ```
- If you wish to modify the sample, run the change mode command:
  ```
  chmod +w *
  ```

Build the Samples

Samples are shipped in a pre-built state. They are ready to run. All samples are built in 32-bit mode.

All samples contain the following files:

- project file
- configuration file
- makefile

The project file and configuration file use the incremental compiler and generate a code store. The makefile uses the batch compiler.

You can build the Core samples, the ones that do not contain User Interface or 2D Graphics classes, in 64-bit mode. The appropriate environment variable or compiler option must be used to build these samples in 64-bit mode. See the topic "Build a 64-Bit Enabled Application" on page 24 for more information.

The samples are built in optimized mode. You can build the samples in debug or static mode using the appropriate compiler options.

The LANG environment variable must be set to build the samples. This value is used to locate the resource and help files associated with the sample. If you receive build errors when compiling the samples, make sure that your LANG environment variable is correct and that the subdirectory corresponding to the value you are using exists in the sample's directory. For example, the English resource and help

files are found in the en_US subdirectory for each sample and the LANG environment variable must be set to en_US respectively.

The following samples use a mixture of IBM Open Class and ANSI STL classes:

- animals
- evenodd
- intkyset
- letterdq
- planets
- pushpop
- wordseq

AIX

The samples are built using English resources. You can rebuild them in the other supported languages by installing the appropriate filesets, setting the LANG environment variable, and rebuilding the sample.

Build ja_JP Samples

Sample resources are not shipped for ja_JP. You can build these resources and rebuild the sample by following these steps:

1. Install the language filesets for Ja_JP sample resources.
2. Copy the sample to a working directory (as described above in "Copy Samples").
3. Change to the sample's working directory and create a ja_JP subdirectory.
4. Copy all files from the sample's Ja_JP subdirectory to the ja_JP subdirectory.
5. Change to the ja_JP subdirectory and run the following command on each file (where *filename* is the name of the file and *outfilename* is the name of the output file):

   ```
   iconv -f IBM-932 -t IBM-eucJP filename > outfilename
   ```

6. Set the LANG environment variable as follows:

   ```
   export LANG=ja_JP
   ```

7. Rebuild the sample.

Execute a Sample

We recommended that you change to the sample's directory when running the sample. Some of the samples look for help or data files in their current working directory.

RELATED CONCEPTS

IBM Open Class Overview
"IBM Open Class Applications" on page 4

RELATED REFERENCES

"Build a 64-Bit Enabled Application" on page 24

Obsolete or Ignored Member Functions

The following sections define obsolete and ignored functions and explain how to use these functions in the IBM Open Class. To develop portable applications, you should be aware of these areas.

The IBM Open Class constantly evolves to improve quality and design. As a result, some functions and classes become obsolete or ignored. A set of macros identifies these obsolete functions and classes so you can migrate to replacements.

You can find these macros in the icomdefs.h header file. One macro conditionally defines the obsolete level for the current version of the library by the platform you are using. Another set of macros defines the obsolete level for previous versions. The following excerpt from icomdefs.h defines the first obsolete level as 310:

```
#define IC_OBSOLETE_1    310
#define IC_OBSOLETE_2    400
#define IC_OBSOLETE_3    410
#define IC_OBSOLETE_4    500
// ———————————— Obsolete Levels ————————————
#ifndef IC_OBSOLETE
    #ifdef IC_WIN
        #define IC_OBSOLETE  400
    #endif
    #ifdef IC_PM
        #define IC_OBSOLETE  400
    #endif
    #ifdef IC_AIX
        #define IC_OBSOLETE  400
    #endif
    #ifdef IC_400
        #define IC_OBSOLETE  310
    #endif
```

An obsolete interface is then wrapped as follows:

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

Notice that IC_OBSOLETE is conditionally defined in icomdefs.h so that you you can set its value. You can easily identify the obsolete interfaces you are currently using by defining IC_OBSOLETE to be greater than any of the obsolete levels. For example, if you define IC_OBSOLETE to be 510, you will receive compile errors for each obsolete function used in your code.

There are several important guidelines regarding obsolete functions:

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

Even if you get no error messages, code compiled with one of these macros defined cannot be run because the generated code does not match any shipped dynamic or static libraries on your system. These macros are only provided to assist you in identifying obsoleted functions and code must be recompiled without any of these macros defined to be executable.

Ignored Functions

Ignored functions are functions that cannot be implemented on a particular platform but you still can call from your program.

Generally, these functions are implemented to do nothing. Although their missing functionality should not be critical to the running of most programs, you should examine how you use ignored functions for each platform on which you develop.

You can identify the functions ignored on a specific platform using the following macros. Compiling with these macros will generate a compiler error for each ignored function you use:

Symbol	Platform
IC_PM_FLAGNOP	OS/2
IC_WIN_FLAGNOP	WIN
IC_MOTIF_FLAGNOP	AIX
IC_PMWIN_FLAGNOP	OS/2 WIN
IC_MOTIFPM_FLAGNOP	AIX OS/2
IC_MOTIFWIN_FLAGNOP	AIX WIN

By defining one of these macros, you can identify the functions that have little to no effect on the corresponding platforms. You can check for ignored functions on any platform; you are not limited to only the platform on which you compile.

For example, IFont::setFontShear() is an ignored function under Motif. The following is an excerpt from ifont.hpp:

```
#ifndef IC_MOTIF_FLAGNOP
 virtual IFont
  &setFontShear( const IPoint& point,
                 const IPresSpaceHandle& presSpaceHandle = IPresSpaceHandle() );
#endif
```

By compiling your with IC_MOTIF_FLAGNOP, IC_MOTIFPM_FLAGNOP, or IC_MOTIFWIN_FLAGNOP defined, the compiler will identify the calls you make to IFont::setFontShear() and any other functions ignored on AIX.

AIX

For example, to find the ignored functions that only AIX will ignore, compile your code as follows:

```
icc -Gm+ -dIC_MOTIF_FLAGNOP sampcode.cpp
```

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

Even if you get no error messages, code compiled with one of these macros defined cannot be run because the generated code does not match any shipped dynamic or static libraries on your system. These macros are only provided to assist you in identifying ignored functions and code must be recompiled without any of these macros defined to be executable.

RELATED CONCEPTS

IBM Open Class Overview

RELATED TASKS

"Compile and Build Open Class Applications" on page 22

Chapter 2. Changes in Version 5 of Open Class

This section describes the major changes in version 5 of the IBM Open Class Library. It can help you identify areas in your application that may need to be changed as you migrate your applications to the version of IBM Open Class shipped with VisualAge C++.

This version of VisualAge C++ uses version 5 of the IBM Open Class. See Changes in Version 4 of IBM Open Class to identify changes between versions 3 and 4 of IBM Open Class.

All changes to IBM Open Class are cumulative in the next higher release unless otherwise stated.

General
The following changes apply to IBM Open Class as a whole or apply to more than one component:

- **Boolean Values**
  The IBM Open Class recognizes bool, true, and false as C++ keywords. You will have to update your compiler options accordingly (remove the nokeyword option for these keywords).
- **Building the IBM Open Class Source**
  You can build the IBM Open Class from its source for debugging purposes.
- AIX **Import Libraries**
  To build your applications, you must link with the IBM Open Class import libraries:
  - For dynamic linking, you only need libioc.a.
  - For static linking, use libiocns.a for the base classes, libiocclns.a for the collection classes, and libiocuins.a for the user interface classes.
- **Unicode**
  You can write Unicode applications with the IBM Open Class. All components support UTF-8. The core classes and the 2D classes also support UCS-2. The IBM Open Class has been updated to Unicode 3.0.
- AIX 64-Bit Support
  You can write 64-bit applications with the core classes on AIX 4.3.
- **Version Number**
  The macro IC_MAJOR_VERSION is set to 500 and IC_OBSOLETE is set to 400 for IC_PM, IC_WIN, and IC_AIX. The macro IC_MINOR_VERSION is set to 0 and is updated as program fixes are manufactured and released. You can find these values in the icomdefs.h file.

Application Control

- **IThread::start**
  This function creates either a INonGUIThread or a GUI-based thread determined by IThread::defaultAutoInitGUI. The default flag can be set with a call to setDefaultAutoInitGUI(bool initFlag = true). This restores behavior of the class that existed prior to version 4 of IBM Open Class.
- **IThreadFn**
  The IThreadFn object is now referenced counted. You can use the functions

addRef and removeRef to extend the lifetime of your IThreadFn object or share the same IThreadFn object among multithreaded thread objects.

Collection Classes

- **Standard Template Library**
  You can use the ANSI Standard Template Library (STL) containers in an IBM Open Class application. We recommend that you use the STL instead of the IBM Collection classes for new development.
- AIX **Import Libraries**
  The import library libiocclns.a contains the IBM Collection classes.
- **Wrapper Classes**
  Previous collection wrapper classes that were shipped in the IBM Open Class source code (in the source/core/collwrap directory) but were not built into the libraries have been removed. Applications that used the undocumented i*2.h headers (for example iset2.h) will need to be changed to use the corresponding header without the "2" in its name (for example iset.h).

Text and International Classes

- **Unicode 3.0**
  The IBM Open Class Library has been updated to the Unicode 3.0 standard. Many character attributes have been updated since Unicode 2.0 (the supported standard in Version 4 of IBM Open Class). The enumerated types in the Unicode Classes have also been updated to the new standard. You might need to update your program to cope with this change.
- **ioc::unichar_t Data Type**
  A more flexible data type called ioc::unichar_t is now used instead of UniChar. For compatibility reasons, this new data type has the same definition as UniChar (unsigned short). However, if you define the macro __IOC_USE_WCHAR, the definition of ioc::unichar_t is changed to wchar_t. This is convenient if you compile IBM Open Class applications on operating systems with native support for Unicode in their C runtimes.
- **IUnicode::EUnicodeScript**
  The constant kPrivateUse from the enumerated type IUnicode::EUnicodeScript has been replaced by kPrivateUseArea.
- WIN OS/2 **Default Locale**
  IDate, ITime, ITimeStamp, IString, and IBuffer all assume C runtime locale by default. Users need to explicitly call IString::disableInternationalization() to restore to the original behavior. (The original behavior is to not use C runtime locale by default.)
- **National Language Support**National language support is on by default. The following statements turn off national language support:

  WIN OS/2
  ```
  SET ICLUI_I18N=OFF
  ```

  AIX
  ```
  export ICLUI_I18N=OFF
  ```
- **IMessageText**
  The return type for IMessageText::messageFile has been changed from char* to const char*. The return type for IMessageText::successful has been changed from int to bool.

Streams and Storage

- **ANSI 3.0 Stream Library**
  The IBM Open Class supports the ANSI 3.0 stream library. You can use both new and old stream libraries as long as the ANSI stream classes explicitly use the std namespace.

- **Release to Release Data Compatibility**
  To support Release to Release Data Compatibility (RRDC) streaming for future releases, the streaming model for the following classes have been changed. This change breaks streaming compatibility between Version 4 and Version 5 of IBM Open Class:
  - Base Library:
    - ITabStop
    - IBinaryCodedDecimal
    - IDate
    - IString
    - ITime
    - ITimeStamp
    - IVersion
  - 2D Graphics Library
    - IGPoint2D
    - IGRPoint2D
    - IGRect2D
    - IGPolygon2D
    - IGArea
    - IGCurve2D
    - IGImage
    - IGLine2D
    - IGPolyline2D
    - IGrafMatrix
    - IGQuadrilateral
    - IRawArray

User Interface Classes

- **IViewPort Clipping Window**
  IViewPort::clippingWindow returns the clipping window of a view port. Previously you had to use one of the following to get this window:
  - the window identifier IC_VIEWPORT_VIEWRECTANGLE
  - the window handle returned by IViewPort::handleForChildCreation

  IViewPort::setViewWindow throws an exception if the specified view window is not parented by the clipping window. Previously this condition was not detected until the view port ran its layout routine.
- **IBaseComboBox Obsoleted Style**
  The style IBaseComboBox::border3D is now obsolete. It had no affect on any platforms.
- AIX **Font**
  The AIX-only function IFont::fontSet returns void* instead of _XFontSet*. You should cast the return value of this function to XFontSet. (The typedef XFontSet in Xlib.h has changed in Xlib specification 6).
- **Build with the Motif 1.2 Compatibility Library on AIX 4.3**
  The Open Class Libraries are built using Motif 1.2 and X11R5. On AIX 4.3 the operating system has made Motif 2.1 and X11R6 the default. A user interface application that uses IBM Open Class (and thus Motif 1.2) cannot use Motif 2.1. These two levels of Motif are not binary compatible.

Thus for a user interface application to mix native X-widgets and IBM Open Class widgets, on AIX 4.3 that application will have to compile and link to the *Motif 1.2 compatibility library*. To use these libraries read the information in the /usr/lpp/X11/readme file. You will find information about installing the fileset X11.compat.adt.Motif12 and how to modify the compile/link command to link to the Motif 1.2 compatibility library.

- **IIconControl**
  The following function has been obsoleted:
  IIconControl& IIconControl::setIcon(ISystemPointerHandle::Identifier icon)
  It has been replaced with this function:
  IIconControl& IIconControl::setIcon (ISystemPointerHandle::EPointerType icon)
  If you have overridden a previous version of IIconControl::setIcon, you will have to add a new override that will accept an ISystemPointerHandle::EPointerType enumerated type.
- **IListBox**
  IListBox can now sort items in a locale sensitive manner. It relies on the internationalization setting of IString.
- **IToolBarButton Styles**
  In Version 4 of IBM Open Class, using the styles IToolBarButton::useIdForBitmap or useIdForText when creating toolbar buttons with window IDs in the IBM Open Class reserved range (greater than or equal to IC_ID_BASE) causes IToolBarButton to load the bitmap and text for the button only from the IBM Open Class resource library. In Version 5 of IBM Open Class, IToolBarButton first tries to load the bitmap and text for the button from the application resource library. If not found, IToolBarButton loads these resources from the IBM Open Class resource library.

  In some cases, toolbar buttons whose text and/or bitmap were supplied by IBM Open Class may now show text and/or a bitmap from the application resource library. If this is not desired, you can change the application in one of two ways:

  1. Change the ID of the string in the string table or the bitmap resource to avoid the range of IBM Open Class reserved toolbar button IDs.
  2. Replace use of the styles IToolBarButton::useIdForBitmap and/or useIdForText with calls to IToolBarButton::setBitmap and/or the inherited setText to load these resources from the IBM Open Class resource library.

Error Handling and Testing Classes

- **Non-GUI Exceptions**
  The user interface exception classes have been moved from the core library to the UI library. As a result, your non-GUI applications should include the file "iexcbase.hpp" rather than "iexcept.hpp" (which contain the GUI exceptions). This has no effect to GUI applications.
- **Deprecated Operators**
  The const char* and implicit const char* operators are now deprecated. Use either text() or textW() to retrieve exception text (or message text) in either 8-bit char form or 16-bit ioc::unichar_t form.
- **ANSI Exceptions**
  The IBM Open Class uses the ANSI class library extensively. You may want to catch both ANSI exceptions and IBM Open Class exceptions.
- **baseLibrary Error Code Group**
  This error code group has been changed from "IBM Class Library" to "IBM Open Class Library."
- **IMessageText**
  Constructing an IMessageText object with uncasted NULL pointers for file name

and message parameters can now cause an ambiguity error at compile time. You should cast these NULL pointers to either const char* or const ioc::unichar_t*.

RELATED CONCEPTS

Changes in Version 4 of the IBM Open Class

Chapter 3. Changes in Version 4 of IBM Open Class

This section describes the major changes in version 4 of the IBM Open Class Library. It can help you identify areas in your application that you may have to change as you migrate your applications to the version of IBM Open Class shipped with VisualAge C++.

The current version of IBM Open Class in this version of VisualAge C++ is version 5. See the section Changes in Version 5 of the IBM Open Class to identify changes between Version 4 and version 5 of IBM Open Class.

All changes to IBM Open Class are cumulative in the next higher release unless otherwise stated.

Inheritance
Classes that inherit from either IBase or IVBase no longer exist.

Predefined Constants
Constants that were previously defined in ibase.hpp are now defined in icconst.h.

2D Graphics Classes
The 2D graphics classes provided in VisualAge for C++ Version 3.0 and 3.5 have been replaced by new graphic classes.

Color
The existing IColor class incorporates new color support:

- The enums IColor::Color and IColor::SystemColor have been deprecated in favor of new enums called IBaseColor::EPredefinedColor and IColor::ESystemColor.
- The enum values have been changed from “white” to “kWhite”, and so on.
- If you construct an IColor object with the ESystemColor enum, calling IColor::writeToStream causes the enumerator value to be streamed. Use of the enum is not portable across systems or display devices. The platform makes a best effort to interpret the value appropriately.
- To permit chaining, the upwardly compatible function IColor& setRed(CharIntensity) replaces the function void setRed(CharIntensity).

The IBaseColor class replaces IBasicColor. IColor is now a subclass of IBaseColor. IBasicColor still exists for backward compatibility.

Threads
A non-GUI thread class called INonGUIThread has been added with the following thread priority functions:

- INonGUIThread::threadPriority
- INonGUIThread::setThreadPriority

These replace the following functions:

- IThread::priorityLevel
- IThread::priorityClass
- IThread::setPriority
- IThread::adjustPriority

Scheduling in this release uses the Windows NT® model: processes can only have a priority class, and threads can only have a priority level. Thus, the process automatically sets the priority class for all of its threads, but each thread can set/reset its own priority within the process. Both process priorities and thread priorities are specified as enumerated types.

Process scheduling uses the following enum:

enum INonGUIApplication::EProcessPriority

Thread scheduling uses the following enum:

enum INonGUIThread::EThreadPriority

OS/2

These functions have been moved to INonGUIThread and made OS/2-only.

The following table lists other thread classes that are new or extended in this release:

Class	Description
IExternalProcess	Basic facilities for starting and controlling the execution of processes
IThreadLocalStorage	Provides a portable way to create and access per-thread global data
IEnvironment	Specifies the execution environment for use with a process
ICondition	A class to be used for thread synchronization. When used in conjunction with IResource, provides the classic "Monitors and Conditions" construct
IPrivateCondition	Concrete subclass of ICondition, for use within a process
ISharedCondition	Concrete subclass of ICondition, for use between processes on a single computer

Reference Counting and Thread Functions
The IRefCounted class is now obsolete. The IMRefCounted class has replaced it.

IMRefCounted is a public base class that can be used by any class that needs reference-counting semantics, similar to the IRefCounted class provided by the IBM Open Class library in the past. IMRefCounted differs from IRefCounted in two ways

- IMRefCounted is thread safe. That is, its addRef, removeRef, and count member functions can be called from multiple threads simultaneously without causing data corruption.
- When an IMRefCounted object is created, its initial reference count is set to 0 rather than 1, which is what IRefCounted uses. An initial count of 0 makes IMRefCounted work much more cleanly with smart pointer classes such as ICountedPointerTo.

One result is the removal of IRefCounted as the base class of IThreadFn. Consequently, code that called addRef or removeRef on an IThreadFn object no longer compiles.

If your application uses reference counting to prolong the lifetime of the IThreadFn object beyond that needed by the IThread or INonGUIThread object running it, then you must change your code to create a new IThreadFn object for each IThread or INonGUIThread that needs it.

Boolean Type Definition
Boolean and IBoolean have been changed to bool to meet ANSI standards.

Bidirectional Language Support
Bidirectional (BIDI) language support has been added. Classes and functions that support a BIDI environment include the following:

- IBidiSettings::applicationDefaults static function
- IBidiSettings::setApplicationDefaults static function
- IWindow::leftToRight style
- IWindow::rightToLeft style
- WIN OS/2 IWindow::setBidiSettings protected virtual function

Collection Classes
These classes now support interest-based notification and streaming.

Text and Internationalization Classes
These classes are new in this release. It provides support for creating internationalized applications that handle text expressed in various encoded character systems. Using these classes, you can transcode text expressed in supported character encoding systems to and from Unicode. IText class provides support for Unicode data, and transcoder classes provide the ASCII to Unicode conversion. Collation is also supported. The primary classes provided by the Text Framework are:

- IText, a variable-length styled string class you can use for storing styled or unstyled international text.
- ITextBoundary, which implements methods for locating boundaries of characters, words, lines, and sentences.
- Style classes, which can be applied to individual characters, ranges of characters, or paragraphs.
- Iterator classes, which provide access to the character data in IText objects.
- Collation classes that support comparing Unicode string in a cultural-sensitive manner.
- Transcoder classes that support conversion of a string from one codepage to another.

The is*xxx* functions (for example isDigits()) from IString return false for empty string. This is a change in behavior from Version 3 of C Set++ for AIX.

Notification Classes
These classes have been extended with the following features:

- A filtering mechanism so that the client object can specify what types of notification to receive
- Asynchronous, in addition to synchronous, notification

Application Control
The Application Control classes have the following enhancements:

- Distinction between non-GUI application (INonGUIApplication) and GUI application (IApplication)
- New DLL loading mechanism allows loading DLLs from the user's NLSPATH

Test Classes
The Test classes are new in this release. These classes include the following features:

- ITest abstract base class provides a standard testing API.
- ITestCollection is a collection of ITest objects and allows them to be run sequentially.
- ITestMultiplexer allows multiple decision functions applied to a single test target.
- ITimingTest can measures the time it takes to perform a specific operation.

File Systems
The File System classes are new in this release. It gives users access to all file system objects such as volumes, directories, files, file contents and operations in a platform-independent manner. The following are typical tasks you can perform using the File System classes:

- Creating, moving, copying, deleting, locating, and manipulating file system objects.
- Manipulating and parsing pathnames.
- Accessing the attributes of files, directories, and volumes.
- Accessing the contents of volume, directory, and file objects using the IBM Open Class stream classes.
- Iterating through directories and volumes.

Streaming
Capabilities supported by the Streaming classes include monomorphic streaming, polymorphic streaming, platform-to-platform data compatibility (a document streamed out on one platform will be readable on another platform), and release-to-release data compatibility (a document written by one version of an application or system will be readable by a newer or an older version).

RELATED CONCEPTS

"AIX Changes in Version 4 of the IBM Open Class"
"Changes in Version 4 of the IWindow Class" on page 46
"Changes in Version 4 of the Handler Classes" on page 50
"Changes in Version 4 of the Canvas Classes" on page 53
"Changes in Version 4 of the Toolbar Classes" on page 56
"**Other Changes in Version 4 of the IBM User Interface Classes**" on page 58
Changes in Version 4 of the IBM 2D Graphics Classes
"Changes in Version 4 of the IBM Collection Classes" on page 72
"Deprecated Functions in Version 4 of the IBM Open Class" on page 73
Changes in Version 5 of the IBM Open Class

AIX Changes in Version 4 of the IBM Open Class

User Interface Classes
These classes support the following features:

- Accelerators: Support for accelerators existed in the previous AIX release, although in a more limited form.
- Clipboard
- Direct Manipulation
- Fly-Over Help
- Resources
- Toolbar
- Handling of Mouse Movement Events
- Portable Font Dialog
- 2D Graphics
- Timer Support
- Bidirectional (BIDI) Support

Restricted Motif Support for IMouseHandler::mousePointerChange
This function is now supported on all platforms, although its use on Motif is restricted. For this callback function to be called each time the mouse is moved, you must create the IMouseHandler with the style IMouseHandler::allMouseMoves.
If you remove a handler overriding IMouseHandler::mousePointerChange, or that handler does not handle the pointer-change event, the mouse pointer is not automatically reset to the default pointer. Instead, when the mouse is over the window it uses the last pointer set by the handler.

Alternatively, you may change the mouse pointer by calling IWindow::setMousePointer. To change the mouse pointer dynamically based on what portion of the window the mouse is over, override IMouseHandler::mouseMoved to call IWindow::setMousePointer (mouseMoved is the best place to determine whether the mouse changed and the mouse's location). You could get inconsistent behavior if you use both IWindow::setMousePointer and an IMouseHandler to change the mouse pointer for a window, depending on the order that you call setMousePointer and attach the handler (and whether the handler returns true or false).

Toolbars on Motif
The Open Class Library now contains IToolBar, IToolBarButton and the other toolbar related classes with limited functionality. The following Motif toolbar functionality is provided at this time:

- The ability to create a toolbar at any location: above, below, left and right of the client or as a floating toolbar.
- Support for multiple toolbars in same location.
- Addition of any window to the toolbar.
- Bitmap view and text view for buttons.
- Toolbar filtering.
- IToolBar::setLocation to move a toolbar from one location to another. Due to Motif system restrictions, certain toolbar movements are restricted. (A new function IToolBar::isMoveValid has been created to verify toolbar movements.)
- Toolbar container.

The following Motif toolbar functionality has not been implemented at this time:

- IToolBar::bitmapAndText view with only text
- Custom drawing through IToolbarButton and ICustomButton virtual functions
- ICustomButtonDrawEvent

- Button latching and disabled buttons
- Drag/drop of toolbar or buttons

RELATED CONCEPTS

"Chapter 3. Changes in Version 4 of IBM Open Class" on page 41
"Changes in Version 4 of the IWindow Class"
"Changes in Version 4 of the Handler Classes" on page 50
"Changes in Version 4 of the Canvas Classes" on page 53
"Changes in Version 4 of the Toolbar Classes" on page 56
"**Other Changes in Version 4 of the IBM User Interface Classes**" on page 58
Changes in Version 4 of the IBM 2D Graphics Classes
"Changes in Version 4 of the IBM Collection Classes" on page 72
"Deprecated Functions in Version 4 of the IBM Open Class" on page 73

User Interface Class Changes

Changes in Version 4 of the IWindow Class

Window Owner versus Parent
The use of an owner different from the parent is discouraged in child windows. In top level windows (IFrameWindow objects, typically) owner and parent remain distinct and are supported portably. A top level window owned by another top level window remains above its owner in the Z order and is closed when its owner is closed. Modality also depends on the owner. In IFrameWindow, the parent window is the one that is less portable. An IFrameWindow usually has a parent of zero or IWindow::desktopWindow, which is the portable variant. Setting the parent of an IFrameWindow to another window can be useful in some scenarios but is less portable. In particular, an IFrameWindow child of another IFrameWindow behaves differently on the three platforms.

The API perspective has no change. Constructors which took both a parent and an owner continue to do so, even if the owner is ignored. This product does not provide parent-only-required constructors. In essence, the change here is one of documentation. The additional parameters are not needed and are ignored. However behavioral changes may occur on OS/2 if specific behavior was expected when the parent was different from the owner.

Changes to IObjectWindow and IWindow for Support of "bound" Windows
In previous releases, the notion of a primary window as defined by the operating system, and the concept of a window requiring that the message queue persist for the lifetime of that window, were treated as a single concept. Both concepts are useful, and they are now treated as distinct in the interface, as follows:

- The behavior of IWindow::setParent and setOwner has changed. IWindow::setParent and setOwner have changed so their checks for setting the primary window flag matches IWindow::startHandlingEventsFor. This breaks any code that relies on using IWindow::setParent to remove the primary window flag. A specific interface for being able to control the primary window flag has also been added.
- The behavior of IWindow::isPrimary has been changed so that it only returns true if the window is truly a primary window.
- A new style has been be added to IObjectWindow, boundToMessageQueue. The default style is boundToMessageQueue.

- A new protected function has been added to IWindow to bind the message queue. The IObjectWindow constructor must call this function if the style is present.
- The three places that the primary flag can be set (setParent, setOwner, and startHandlingEventsFor) should also set the "bound" flag as appropriate. All primary windows are also "bound" windows, but all "bound" windows are not primary.
- "Bound" windows are now checked instead of primary windows when deciding whether to shut down the message queue.
- The following are the new interfaces for IObjectWindow and IWindow:

```
/* New interface in IObjectWindow */
public:
  class Style;
  IObjectWindow ( const IObjectWindow::Style& style = defaultStyle() );
  INESTEDBITFLAGCLASSDEF0( Style, IObjectWindow );
    static const Style
      IC_IMPORTU classDefaultStyle,
      IC_IMPORTU boundToMessageQueue,
      IC_IMPORTU noStyle;
    static Style
      defaultStyle ( );
  static void
    setDefaultStyle ( const Style& style );
private:
  INESTEDBITFLAGCLASSFUNCS(Style, IFrameWindow);
    static Style
      fCurrentDefaultStyle;
/* New interface in IWindow */
protected:
  IWindow
    &bindMessageQueue ( );
  bool
    isBoundToMessageQueue ( ) const;
```

In the canvas classes collections of objects are created that are nothing more than a pointer to an IWindow with a bit of extra data for the class that is specific to the IWindow. These collections are needed because there is no way to extend the data of an IWindow from outside the object. These collections of objects must be kept up to date so that they accurately represent the child windows of a parent object in sibling order. This requirement is difficult enough that ISetCanvas rebuilds the collection each time it does layout so that it is accurate.

Extensions to the data of an IWindow are now allowed. IWindow is updated to store an array of pointers to objects. Any class that needs to store data in an IWindow object first queries to receive a *magic token*. A class uses a magic token to to store and retrieve data from an IWindow object. (The magic token is implemented as an index into the array of pointers.) Instead of creating collections of objects to represent additional data, classes like the canvases define and create an object that they store as additional IWindow data. They then use IWindow::ChildCursor to walk through the IWindow children and retrieve their extended data. Support has been added to IWindow::ChildCursor to allow it to visit only IWindow children.

The following classes and members have been implemented:

Class or Member	Description
IWindow::DataHandle	This is the magic token.

Class or Member	Description
IWindowData	This abstract base class of data added to an IWindow provides a virtual destructor so that IWindow can manage the destruction of the object. IWindow adopts the object and deletes it in the IWindow destructor.
static DataHandle IWindow::dataHandleWithKey (const char* dataKeyName);	This thread-safe function increments a counter so that each new dataTypeName gets a different offset. It also keeps a collection of registered names so that duplicate requests return the same DataHandle.
inline IWindowData* IWindow::windowData (const DataHandle& typeToken) const;	This inline function indexes the array (stored directly in IWindow) and returns the pointer at the offset of the passed DataHandle.
IWindow& IWindow::adoptWindowData (const DataHandle& typeToken, IWindowData* windowData);	This member checks to see if an array of IWindowData exists for the window and allocates it if there is none. It also checks to see if the array is large enough to contain the index and reallocates it if necessary. It then stores the pointer in the array. If an entry already exists at the handle location, it is deleted before the new window data is stored.
IWindow::ChildCursor::ChildCursor (IWindow& parent, bool onlyIWindowChildren = false);	This function is updated to allow iteration only over IWindow children. When cursorIWindowChildren is true, the cursor movement functions call windowWithHandle to query and store the IWindow* at the cursor location. The cursor movement function continues to enumerate and call windowWithHandle until all children have been visited or windowWithHandle returns a valid IWindow pointer.
IWindow* IWindow::childWindowAt (const ChildCursor&) const;	This function returns the IWindow* stored in the child cursor. The function first checks the validity of the cursor (IASSERTSTATE(fWindowHandle != 0)). If the child cursor's IWindow* value is not zero, it returns that value. If the cursor's IWindow* value is zero, it returns the result of windowWithHandle (which may be zero, no exception is thrown).

There is a new version of IWindow::create that takes an IWindow* for parent and owner instead of IWindowHandles. Use this new create function when you inherit from an IWindow object. The library then ensures that the correct handle for the creation (parent->handleForChildCreation) of the control is used.

The use of handle on Motif was different from its use on Windows and OS/2. If an aggregate could have children, such as IFrameWindow and IViewPort, handle was used as the parent of the child. Windows and OS/2 get around this problem by using a setParent to change the parent of the child to the correct control. Motif cannot use this technique because reparenting is not available. Motif had a

topHandle and a handle so that children of a control can have a different parent handle than the handle used for geometry management. In Motif, the topHandle was the handle of the control (widget) that encompassed all of the widgets of the aggregate. The topHandle was used for geometry management of the control such as moving and sizing the control. The problem with topHandle is that it was used only on Motif and had the same meaning as handle does on OS/2 and Windows.

A new member function returns the handle that a child of a control should use as their parent. When a control is created and an IWindow* is specified as the parent, handleForChildsParent is used instead of handle. The default implementation of handleForChildsParent returns just handle. Any subclass that needs a different implementation, like IViewPort, overrides this member function and returns the appropriate handle.

Here are the changes to IWindow:

```
#ifdef IC_PMWIN
  virtual IWindowHandle
    create ( unsigned long id,
             const char* text,
             unsigned long style,
             const char* windowClass,
             const IWindowHandle& parent,
             const IWindowHandle& owner,
             const IRectangle& initRect,
             const void* ctlData,
             const void* presParams,
             IWindow::SiblingOrder ordering = defaultOrdering(),
             unsigned long extendedStyle = 0 );
  virtual IWindowHandle
    create ( unsigned long id,
             const char* text,
             unsigned long style,
             const char* windowClass,
             const IWindow* parent,
             const IWindow* owner,
             const IRectangle& initRect,
             const void* ctlData,
             const void* presParams,
             IWindow::SiblingOrder ordering = defaultOrdering(),
             unsigned long extendedStyle = 0 );
#endif
#ifdef IC_MOTIF
  virtual IWindowHandle
    create ( unsigned long id,
             const char* text,
             unsigned long style,
             IXmCreateFunction createFunction,
             const IWindowHandle& parent,
             const IWindowHandle& owner,
             const IRectangle& initRect,
             const void* callerArgList,
             const unsigned int callerNumberArguments,
             IWindow::SiblingOrder ordering = defaultOrdering(),
             unsigned long extendedStyle = 0 );
  virtual IWindowHandle
    create ( unsigned long id,
      const char* text,
             unsigned long style,
             IXmCreateFunction createFunction,
             const IWindow* parent,
             const IWindow* owner,
             const IRectangle& initRect,
             const void* callerArgList,
```

```
                const unsigned int callerNumberArguments,
                IWindow::SiblingOrder ordering = defaultOrdering(),
                unsigned long extendedStyle = 0 );
#endif
```

The implementation of the new create function uses the old create function and passes through the correct handles:

```
// IWindow::create()
// This create will take care of the using
// the correct parent's handle for
// the creation of children.
IWindowHandle IWindow::create( unsigned long id,
   const char* text,
unsigned long style,
IXmCreateFunction createFunction,
const IWindow* parent,
const IWindow* owner,
const IRectangle& initRect,
const void* callerArgList,
const unsigned int callerNumberArguments,
IWindow::SiblingOrder ordering,
unsigned long extendedStyle )
{
  return this->create( id,
                       text,
                       style
                       createFunction,
                       parent->handleForChildsParent(),
                       owner->handle(),
                       initRect,
                       callerArgList,
                       callerNumberArguments,
                       ordering,
                       extendedStyle );
}
```

RELATED CONCEPTS

"Chapter 3. Changes in Version 4 of IBM Open Class" on page 41
"AIX Changes in Version 4 of the IBM Open Class" on page 44
"Changes in Version 4 of the Handler Classes"
"Changes in Version 4 of the Canvas Classes" on page 53
"Changes in Version 4 of the Toolbar Classes" on page 56
"**Other Changes in Version 4 of the IBM User Interface Classes**" on page 58
Changes in Version 4 of the IBM 2D Graphics Classes
"Changes in Version 4 of the IBM Collection Classes" on page 72
"Deprecated Functions in Version 4 of the IBM Open Class" on page 73

Changes in Version 4 of the Handler Classes

IPresSpaceHandle

This is an existing class whose constructor was changed to take an additional optional window handle. The window handle is important when creating IPresSpaceHandle objects on OS/2 to draw with the new 2D graphics classes. Letting the value of the window handle default to zero can cause incorrect drawing when using upper-left coordinates.

New interface on IWindow for Event Dispatching

The following new functions allow event handlers for a window to receive or not receive certain messages have been added:

- IWindow::startHandling(const EventMask& mask)

- IWindow::stopHandling(const EventMask& mask)
- IWindow::isHandling(const EventMask& mask) const

where EventMask is a nested class that derives from IBitFlag. The following are public static event masks that IWindow provides:

- IWindow::EventMask someMouseMoves
- IWindow::EventMask allMouseMoves
- IWindow::EventMask mouseEntersLeaves

On OS/2 and Windows, someMouseMoves and allMouseMoves are equivalent. The style mouseEntersLeaves enables someMouseMoves on certain platforms such as Windows NT. The stopHandling function prevents any handlers from being called for that window to process the specified type of event. Conversely, the startHandling function allows handlers to be called for the event.

The stopHandling function does not prevent IWindow::defaultProcedure from being called. No attempt is made, however, to call private default handlers of a window. Therefore, if you turn off receipt of messages, you may change the default behavior of a window. For example, it can prevent the user from being able to move split bar of an ISplitCanvas on OS/2 or Windows.

To allow multiple calls to startHandling and stopHandling for an event type, the IBM Open class uses reference counting. A default bool argument to stopHandling has been implemented to keep handlers from receiving messages regardless of the reference counting state. By default, handlers of all windows will receive all events on OS/2 and Windows. For performance reasons, Motif handlers will not receive mouse move or mouse enter/leave events by default. Note that an application generally should not need to call these functions. To cover the most common case where these functions need to be called IMouseHandler is enhanced to call them on the application's behalf.

Cases where an application needs to call IWindow::startHandling or stopHandling include the following:

- The application is processing mouse move events in a handler that does not derive from IMouseHandler.
- The application prefers to not modify or cannot modify the call to the IMouseHandler constructor. For example, it uses a mouse handler class provided by a third party.
- The application needs to dynamically start and stop the receipt of mouse messages, which IMouseHandler does not support.

Style Added to IMouseHandler

An optional style argument to the IMouseHandler constructor has been added. These styles dictate the type of mouse messages that the handlers of the window can receive. As a result, they actually affect more than just the single mouse handler. The constructor now looks like the following:

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

where Style is a nested class that derives from IBitFlag. IMouseHandler provides the following the public static styles:

- static const IMouseHandler::Style noMouseMoves
- static const IMouseHandler::Style someMouseMoves
- static const IMouseHandler::Style allMouseMoves
- static const IMouseHandler::Style mouseEntersLeaves

The default style is someMouseMoves. This supports backward compatibility with existing mouse handlers written for OS/2 and Windows by requesting a portion of the mouse move events for the window's handlers. On OS/2 and Windows, someMouseMoves and allMouseMoves are equivalent. The IMouseHandler constructor calls IWindow::startHandling or stopHandling as appropriate (these styles map to the EventMask flags). If the application constructs the handler with the noMouseMoves style, then code is added to IMouseHandler::dispatchHandlerEvent to prevent its mouseMoved virtual function from being called, even if the handler receives a mouse move event. The same rule applies to not specifying the mouseEnters or mouseLeaves style. In this case the new mouseEnter and mouseLeave virtual functions described later are never called.

The following standard auxiliary style members are added to IMouseHandler:

- static Style IMouseHandler::defaultStyle()
- static void IMouseHandler::setDefaultStyle(const Style& style)
- static const Style classDefaultStyle

Pointer Enter/Leave Support
The following protected virtual functions in IMouseHandler are new:

```
virtual bool
  IMouseHandler::mouseEnter ( IEvent& event ),
  IMouseHandler::mouseLeave ( IEvent& event );
```

IMouseHandler calls these virtual functions when the mouse pointer crosses the dispatching window's boundary.

The return value from processing the mouseEnter and mouseLeave virtual functions has no effect on whether the mouseMoved virtual function is called. The same is true the other way around as well. IMouseHandler does not support a time delay (your application can use ITimer to achieve this support).

Make IMousePointerEvent Available on OS/2 and Windows Only
This class is not implemented on AIX, but it is not deprecated either. In the class library it is only used by the deprecated IMouseHandler::mousePointerChange function and the IC_PMWIN-only IMousePointer class.

IMouseConnectionTo Deprecated
This class template has been deprecated and it not implemented on AIX. Replace all use of IMouseConnectionTo with one of the following two classes, as appropriate:

- IMouseClickConnectionTo
- IMouseMoveConnectionTo

IMousePointerHandler
This class performs no processing on Motif, other than to enable the window to receive a limited number of mouse movement events. IMousePointerHandler still overrides mousePointerChange on Windows and OS/2.

RELATED CONCEPTS

"Chapter 3. Changes in Version 4 of IBM Open Class" on page 41
"AIX Changes in Version 4 of the IBM Open Class" on page 44
"Changes in Version 4 of the IWindow Class" on page 46
"Changes in Version 4 of the Canvas Classes" on page 53
"Changes in Version 4 of the Toolbar Classes" on page 56

"**Other Changes in Version 4 of the IBM User Interface Classes**" on page 58
Changes in Version 4 of the IBM 2D Graphics Classes
"Changes in Version 4 of the IBM Collection Classes" on page 72
"Deprecated Functions in Version 4 of the IBM Open Class" on page 73

Changes in Version 4 of the Canvas Classes

The following function was added to ICanvas to determine if windows have been added to the canvas.You may need to override this function in your own ICanvas derived classes.

```
virtual bool
  hasChildrenToLayout() const;
```

New Protected Constructor for ISetCanvas
The following protected functions were added to ISetCanvas to simplify the construction of derived classes and support delayed window creation:

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

Calling the protected constructor does not create a presentation system window. However, you can create a canvas presentation system window by calling initialize. The style arguments are of type unsigned long so that styles defined by a class derived from ISetCanvas can also be used.

IToolBar and IToolBarContainer call this protected constructor, which gives a set canvas with a default configuration but no underlying presentation system window. IToolBar and IToolBarContainer later call ICanvas::initialize to create the presentation system window.

Border Support
ICanvas now has the capability to draw a border with text. Previously, ISetCanvas was the only canvas class with this support. ICanvas provides the following public members to provide this support:

```
static const Style
 ICanvas::border;
static INotificationId const
 ICanvas::borderTextId;

virtual ICanvas&addBorder ( ),
 &removeBorder ( ),
 &setBorderText ( const IText& borderText, bool showBorder=true),
 &setBorderText ( const IResourceId& borderText, bool showBorder=true),
 &setBorderColor ( const IColor& borderColor),
 &resetBorderColor ( );

virtual bool
 hasBorder ( ) const;
IText
 borderText ( ) const;
 virtual IColor
borderColor ( ) const;
```

The ICanvas class demonstrates the following behavior when drawing borders:

- Constructing a canvas with the style ICanvas::border creates a canvas with a border but no text.
- Calling ICanvas::setBorderText(text) adds and displays a border (and the corresponding border style) with text.
- Calling a canvas containing border text with ICanvas::setBorderText(0) causes a canvas to remove its border text, but leaves the border.
- Calling removeBorder removes the border and any border text and repositions its children to use the additional space (the canvas does not draw its border text).

You can query and set the color or font of the border text using the IText class.

The following protected members were added to ICanvas for derived classes to implement layout within a border:

```
IPoint
 topLeftLayoutOffset ( ) const,
 bottomRightLayoutOffset ( ) const;
IRectangle
 rectangleInsideBorder ( const ISize& sizeWithBorder ) const;
ISize
 sizeWithBorder ( const ISize& sizeWithoutBorder ) const;
```

The following member from ISetCanvas was removed:

```
static const Style ISetCanvas::border;
```

The following members of ISetCanvas have changed its behavior::

```
// if specifying text, change to call
// setBorderText(text, true), else
// change to call removeBorder.
ISetCanvas::setText();

// change to call borderText
ISetCanvas::text();
```

The following member of ISetCanvas was deprecated:

```
// Replaced by ICanvas::borderTextId.
static INotificationId const ISetCanvas::textId;
```

Explicitly Adding and Removing Child Windows from ISetCanvas
An interface was added so you can add and remove child windows from the layout of a set canvas.

- A tool bar (which is derived from ISetCanvas) in Motif can now filter misfit controls and remove child controls. It cannot currently do this given the lack of IWindow::setParent.
- An application can temporarily remove a control from the layout of a set canvas in Motif (an application usually does this today in OS/2 by reparenting the control to the object window).
- IToolBar can behave predictably when a control has not explicitly been added to the tool bar.

To simplify layout processing in ISetCanvas, the application should specify at construction time whether it is using the add/remove interface. Use the following style to do this:

```
 ISetCanvas::explicitAddsNeeded
```

The following interface was added so the application can add child windows or remove them from the layout of the canvas:

```
virtual ISetCanvas
 &add ( IWindow* child ),
 &remove ( IWindow* child),
 &replaceChildWith ( IWindow* existingChild IWindow* newChild );
bool
 areExplicitAddsNeeded() const;
```

The ISetCanvas class demonstrates the following behavior when managing child windows:

- For a set canvas to manage the size and position of a child window, that child window must be considered to have been added to the canvas. Not using the style causes all child windows initially to be considered added. Calling the add function to add an already added child window produces no result. Specifying the style causes all child windows initially to be considered not added; in this case you must call add to add a child window to the layout of the canvas. Whether you use the style or not, you can still remove or replace child windows; you can add back removed or replaced child windows. See the isInLayout function for more details on when a child window is considered added or not added.
- When adding a child window, ISetCanvas also shows and enables the child. However, ISetCanvas does not change the z-order of the child window. The z-order will still determine where the child window appears in the layout.
- When removing or replacing a child window, the ISetCanvas hides the child window and sizes it to (0,0). Not hiding the child window causes the child window to remain visible to the user.
- Before replacing a child window, ISetCanvas moves the new child to the existing child's position in the z-order. It copies the tab stop and group styles of the existing child to the new child. It also shows the new child.

To query whether a set canvas is managing the size and position of a child window, the following was added:

```
virtual bool
 isInLayout ( IWindow* child ) const;
```

This function returns true for a child window in these cases:

- If you add the child window to the canvas using a call to ISetCanvas::add or replaceChildWith, and did not remove or replace it afterwards by calling remove or replaceChildWith.
- If you did not create the canvas with the style ISetCanvas::explicitAddsNeeded, and never specified the child window on a call to ISetCanvas::add, remove, or replaceChildWith.

This function returns false for a child window in these cases:

- If you removed or replaced the child window by calling remove or replaceChildWith without later adding it back to the canvas.
- If you created the canvas with the style ISetCanvas::explicitAddsNeeded, never specified the child window on a call to ISetCanvas::add, and never added it to the canvas by calling replaceChildWith.

IViewPort Performance

The following added interface improves IViewPort performance by allowing the viewport child (view window) to be set:

```
virtual IViewPort
 &setViewWindow ( const IWindowHandle& viewWindow );
```

This portable interface boosts IViewPort performance on both OS/2 and Windows by avoiding the child window lookup currently required to find the view window to scroll on each scroll message. This interface also allows you to easily change the window that a view port scrolls.

The restriction of allowing only a single view window per viewport is removed and an exception is no longer thrown when multiple view windows exist. If setViewWindow is not called, the first view window found is used as the viewports' view window. In order to replace a view window with a different view window, you must issue setViewWindow. The previous view window is hidden.

RELATED CONCEPTS

"Chapter 3. Changes in Version 4 of IBM Open Class" on page 41
"AIX Changes in Version 4 of the IBM Open Class" on page 44
"Changes in Version 4 of the IWindow Class" on page 46
"Changes in Version 4 of the Handler Classes" on page 50
"Changes in Version 4 of the Toolbar Classes"
"**Other Changes in Version 4 of the IBM User Interface Classes**" on page 58
Changes in Version 4 of the IBM 2D Graphics Classes
"Changes in Version 4 of the IBM Collection Classes" on page 72
"Deprecated Functions in Version 4 of the IBM Open Class" on page 73

Changes in Version 4 of the Toolbar Classes

New Styles for Direct Manipulation Support

For performance reasons, support for drag and drop is no longer enabled by default. This change necessitated the addition of a new style and deprecation of an existing style. The style IToolBar::dragdrop enables dragging and dropping of the toolbar items. This style is not part of classDefaultStyle. It replaces the style IToolBarButton::noDragDrop, which has been deprecated.

The style IToolBarButton::dragDelete enables the dragging and dropping of toolbar buttons to the shredder; the drag handler queries this style. This style is part of classDefaultStyle. It replaces the style IToolbarButton::noDragDelete, which has been deprecated. Specifying this deprecated style has no effect. Although buttons could originally be dropped on the shredder, to get this style, you must now use the classDefaultStyle or specify the style explicitly.

There is a new override of IToolBar::layout.

A new performance setting, IPerformanceSettings::dynamicToolBarButtons affects the behavior of the following toolbar button functions:

- IToolBarButton::setStandardBitmapSize
- IToolBarButton::setStandardTextLines
- IToolBarButton::setStandardTextWidth

With this setting enabled, these functions dynamically change the settings of all IToolBarButtons created with the IToolBarButton::standardFormat style. Calling these functions causes existing toolbar buttons to adopt the new settings immediately. Each button resizes and redraws with the new settings.

Without this setting enabled, calling these functions causes default settings to be used during the creation of any toolbar buttons that are subsequently created with the IToolBarButton::standardFormat style. This does not affect existing toolbar buttons. By default, this setting is not enabled, which provides for better toolbar performance.

New Default Toolbar Buttons
The new Lotus® toolbar buttons are included in the IBM Open Class resource library.

IToolBar::Style and IToolBarContainer::Style changes
IToolBar::Style and IToolBarContainer::Style has been changed so that you can no longer combine style objects of these types with ISetCanvas::Style objects. If you do specify an orientation or alignment using ISetCanvas::Style objects when constructing an IToolBar or IToolBarContainer, these values are ignored because both IToolBar and IToolBarContainer explicitly set the orientation and alignment themselves. The only ISetCanvas styles that you can set are the packing styles. You can also set the packing styles by calling ISetCanvas::setPackType, and ISetCanvas::decksByGroup which IToolBarContainer gives you by default.

The ability to use IWindow, ICanvas, and ISetCanvas styles on the IToolBar constructor has been deprecated. IToolBar does not pass IWindow styles to the public ISetCanvas constructor it calls, and it ignores extended styles. This is because it explicitly configures the ISetCanvas using function calls. Using application-specified IWindow, ICanvas, and ISetCanvas styles could break existing code. This behavior also applies to IToolBarContainer::Style.

Transparency in Toolbars
To optimize performance, this product no longer supports transparency by default. IToolBarButton::transparentBitmap was added to provide transparency support. Transparent bitmaps are used within toolbar buttons for the following purposes:

- To display the button in an up state. The transparent area of the bitmap shows through to the background color, which can be changed by the user.
- To display a button in latched state. In latched state, a button is drawn in the down state, with the transparent area of the bitmap showing through to a half-toned background look.

To avoid transparency overhead, a new style for IToolbarButtons buttons called IToolBarButton::transparentButton was added. This style is turned off as the default.

For the IBM Open Class supplied bitmaps (IC_ID_OPEN, etc), this product supplies non-transparent bitmaps, and skips any transparency related code. The existing IBM Open Class bitmaps now have new IDs, such as IC_ID_OPEN_WITH_TRANSPARENCY. If you want to use the existing transparency function you must use the new bitmap IDs and set the IToolBarButton::transparentButton style for the button. For user supplied bitmaps, the default is that transparency is turned off. If you want transparency and you supply a transparent bitmap, you must turn the style on for those buttons.

The transparentButton style affects both buttons drawn in an up state and those drawn in a latched state. Without the style set for a button, if the button is drawn in the latched state when the button view is bitmap only, you see the non-transparent latched look. The button is painted as if in the down state, with a slight half-toned look just inside the border and around the bitmap.

When the button view is bitmap and text, the button background color around the text area, which you can configure, might not match the non-transparent gray of the bitmap. In this case, you have the option to use a transparent bitmap for up state and latched buttons. You must make this choice at button creation, or use the function, IToolBarButton::enableTransparency(bool enable = true).

Window Changes
Reparenting of windows is not supported on Motif. This is a Motif-only restriction requiring that any window added to a toolbar must be a child of that toolbar.

OS/2

WIN You can still add previously created windows to a toolbar on OS/2 and Windows since the IBM Open Class will reparent the window automatically.

RELATED CONCEPTS

"Chapter 3. Changes in Version 4 of IBM Open Class" on page 41
"AIX Changes in Version 4 of the IBM Open Class" on page 44
"Changes in Version 4 of the IWindow Class" on page 46
"Changes in Version 4 of the Handler Classes" on page 50
"Changes in Version 4 of the Canvas Classes" on page 53
"**Other Changes in Version 4 of the IBM User Interface Classes**"
Changes in Version 4 of the IBM 2D Graphics Classes
"Changes in Version 4 of the IBM Collection Classes" on page 72
"Deprecated Functions in Version 4 of the IBM Open Class" on page 73

Other Changes in Version 4 of the IBM User Interface Classes

IWindow
Override handleForChildCreation if you have created an aggregate that can have children. Call IWindow::create with the parent window of your aggregate's handleForChildCreation (instead of handle) or use the new IWindow::create override.

AIX

topHandle is replaced with handle; topHandle is deprecated. Do not call addRelated or rely on windowWithHandle working on anything but the registered handle.

Keyboard
The following data types are deprecated, as they provide little utility and are nonportable:

```
public:
  #ifdef IC_MOTIF
    static unsigned long
      IC_IMPORTU ulShiftMask,
      IC_IMPORTU ulAltMask,
      IC_IMPORTU ulCtrlMask;
#endif
protected:
  #ifndef IC_WIN_FLAGNOP
    static const unsigned long
      IC_IMPORTU ulCharacterFlag,
      IC_IMPORTU ulScanCodeFlag,
      IC_IMPORTU ulVirtualFlag,
      IC_IMPORTU ulRepeatFlag,
      IC_IMPORTU ulUncombinedFlag,
      IC_IMPORTU ulShiftFlag,
      IC_IMPORTU ulCtrlFlag,
      IC_IMPORTU ulForCompositeFlag,
      IC_IMPORTU ulCompositeFlag,
      IC_IMPORTU ulInvalidCompositeFlag;
```

```
#endif
static const unsigned long
  IC_IMPORTU ulUpTransitionFlag,
  IC_IMPORTU ulAltFlag;
```

IKeyboardHandler and IKeyboardEvent

WIN You can call the virtual function IKeyboardHandler::characterKeyPress to process WM_KEYDOWN and WM_SYSKEYDOWN messages, in addition to the WM_CHAR messages you could previously process. As a result, code that extracts event data from the IKeyboardEvent using IEvent::parameter1 and parameter2 needs to verify the message type, since the message parameters for WM_KEYDOWN and WM_SYSKEYDOWN differ from those of WM_CHAR. Similarly, IKeyboardEvent::isCharacter can return true now for some WM_KEYUP, WM_SYSKEYUP, WM_KEYDOWN, and WM_SYSKEYDOWN messages, in addition to WM_CHAR messages. You should check the message type if you access data using IEvent::parameter1 and parameter2.

Frame Window

Users who wish only to set the title text on a frame window now have a simple, straightforward, portable way to do this. You no longer have to create an ITitle object to have title text on a frame window.

The following new functions have been added to IFrameWindow:

```
virtual IFrameWindow&
 setTitleText ( const IString& titleText ),
 setTitleText ( const IResourceId& titleResId );
virtual IString
 titleText ( ) const;
```

A notification to IFrameWindow for text changes has also been added:

```
 static INotificationId const
   IC_IMPORTU titleTextId;
```

Fonts

New classes and functions have been added to provide the following functionality:

- Ask for Helvetica 30, such that the font looks the same as in other applications. The font will also scale (on the screen) the same way fonts scale in other applications as device resolution is changed.
- Program in pixels, if you want to.
- Achieve WYSIWYG (metric fidelity), if wanted.

Menus

In previous releases, the IMenu class was derived from IWindow. As a result, you could use an IMenu object to call IWindow functions even though most had no effect on Windows.

For this release, the menu hierarchy has changed. IMenu no longer derives from IWindow. You can only call IWindow member functions by first calling IMenu::window to get the underlying window, under OS/2 and AIX. Selected IWindow functions have been added to IMenu or one of its derived classes and implemented there in a portable way.

IControl

The group and tabStop styles and their associated functions have moved from IControl to IWindow.

New interface for IContainerObject
The following functions set a special in-use icon for this particular object in all containers in which it exists.

- virtual IContainerObject& IContainerObject::setInUseIcon(const IPointerHandle& inUseIcon);
- virtual IContainerObject& IContainerObject::setInUseIcon(const IResourceId& inUseIconId);
- virtual IContainerObject& IContainerObject::setInUseIcon(unsigned long inUseIconId);
- virtual IPointerHandle IContainerObject::inUseIcon( ) const;

The behavior is as follows:

WIN

OS/2 This applies to Windows applications only when they use an IContainerControl object created with the IContainer::pmCompatible style. When the client calls setInUse, the code checks to see if a special in-use icon has been set for the object. If so, this icon is displayed, the original icon cached, and the normal processing continues. This includes sending the container a CM_SETRECORDEMPHASIS message to draw the hatched background behind the icon. When removeInUse is called, the previous icon that was cached will be replaced and the normal processing continues. This includes removing the hatched background. If a special in-use icon is not set for the object, the normal drawing of the hatched background takes place.

AIX

WIN This applies to Windows applications only when they use an IContainerControl object that has not been created with the IContainer::pmCompatible style. When the client calls setInUse, the code checks to see if a special in-use icon has been set for the object. If so, this icon is displayed and the original icon cached. If there is not a special in-use icon set, this call does nothing. When the client calls removeInUse, the previous icon that was cached, if any, is replaced. Do not provide a hatched background because it is an OS/2 (or a Windows container with an IContainer::pmCompatible style) specific behavior.

ICoordinateSystem and Application Origin
To provide the capability to write portable applications, the Open Class Library has defined the following behavior:

- If an application does not call ICoordinateSystem::setApplicationOrientation(), then the orientation defaults to upper-left. This behavior differs from previous releases, where the application orientation was upper-left for Windows and lower-left for OS/2.
- IExtendedRootGrafPort uses the default orientation which you can set with ICoordinateSystem::setApplicationOrientation, unless you specify an orientation when constructing the IExtendedRootGrafPort object.
- Changing the application orientation does not affect existing IExtendedRootGrafPort objects. As a result, calling ICoordinateSystem::setApplicationOrientation in the middle of your application, after IExtendedRootGrafPort objects have already been created, can result in unexpected behavior.
- The enum ICoordinateSystem::Orientation has been deprecated.

- A new enum, ICoordinateSystem::EOrientation() has been created. It has two values: kOriginUpperLeft and kOriginLowerLeft.

Examples:

```
function()
{
  ICoordinateSystem::setApplicationOrientation
                      (ICoordinateSystem::kOriginLowerLeft);
  // The origin defaults to the
  // application orientation
  // lower left
  IExtendedRootGrafPort aPort(hps);
}
function()
{
  ICoordinateSystem::setApplicationOrientation
                      (ICoordinateSystem::kOriginUpperLeft);
  // The origin defaults to the
  // application orientation
  // upper left
  IExtendedRootGrafPort aPort(hps);
}
function()
{
  // The origin defaults to application
  // orientation which defaults
  // to upper left
  IExtendedRootGrafPort aPort(hps);
}
function()
{
  // The origin is always upper left
  // regardless of calls to
  // ICoordinateSystem::setApplicationOrientation()
  IExtendedRootGrafPort
    aPort(hps, ICoordinateSystem::kOriginUpperLeft);
}
function()
{
  // The origin is always lower left
  // regardless of calls to
  // ICoordinateSystem::setApplicationOrientation()
  IExtendedRootGrafPort
    aPort(hps, ICoordinateSystem::kOriginLowerLeft);
}
```

Combo Box

To support toolbars, the current combo box can now drop down past its parent's area. This limitation has been removed by parenting the listbox portion of a drop-down combo box to a shell. This is an AIX-specific enhancement (combo boxes already behave this way on OS/2 and Windows).

Scroll Bar

The IScrollBar class currently contains the autoSize style. This style is supported on OS/2 and AIX, but is not supported in Windows.

AIX

This style was implemented in the class library to always force the scrollbar width (height) to the (hardcoded) system default dimension.

OS/2 The specification of the autoSize style results in a vertical scrollbar that is

automatically sized to the system scrollbar width if a width of zero is specified, or a horizontal scrollbar that will be automatically sized to the system scrollbar height if a height of zero is specified.

The IScrollBar::autoSize style in OS/2 has been deprecated and removed from AIX for the following reasons, which assume a vertical scrollbar:

- In order to use it in OS/2, a width of zero must be specified when the scrollbar is sized. This style, therefore, has no value for a scrollbar in a multicell canvas since the canvas uses the minimum size to determine the optimal width and sizes the scrollbar accordingly. If the scrollbar is not constructed as a child of a multicell canvas, then the system width needs to be used in order to properly position it and other child windows. Whatever width is assumed in order to acquire the preferred layout should actually be used to size the scrollbar.
- The AIX implementation was not equivalent to the functionality provided in OS/2.
- A scrollbar can be constructed using the system scrollbar width as returned by the static IScrollBar::systemScrollBarWidth function.

ICustomButton
IToolBarButton inherits from ICustomButton. In previous implementations, the default protected ICustomButton constructor attached a default ICustomButtonDrawHandler handler. The current implementation of IToolBarButton makes the ICustomButton default handler unnecessary. IAnimatedButton also uses this ICustomButton constructor and does not need the default ICustomButtonDrawHandler. Each toolbar button incurred the overhead of the additional handler attached to it. For performance reasons, this handler has now been removed and the following changes made:

- The protected functions ICustomButton::addDefaultDrawHandler and ICustomButton::removeDefaultDrawHandler have been added.
- The behavior of the ICustomButton protected constructor has been changed so it does not add the ICustomButtonDrawHandler.

You cannot override the default paint handler for custom buttons. Use the two new functions or if the class derived from ICustomButton wants to use the default paint handler, call the addDefaultDrawHandler function to add the handler. This is a change in behavior because a class derived from ICustomButton used to have the default paint handler attached to it automatically. However, a derived class that has its own paint handler derived from the ICustomButtonDrawHandler will not see any behavior changes and might see some performance improvements.

Direct Manipulation
The following functions have been removed because IMenu no longer derives from IWindow:

- IDMHandler::enableDragFrom(IMenuBar*)
- IDMHandler::enableDragFrom(ISubmenu*)

RELATED CONCEPTS

"Chapter 3. Changes in Version 4 of IBM Open Class" on page 41
"AIX Changes in Version 4 of the IBM Open Class" on page 44
"Changes in Version 4 of the IWindow Class" on page 46
"Changes in Version 4 of the Handler Classes" on page 50
"Changes in Version 4 of the Canvas Classes" on page 53
"Changes in Version 4 of the Toolbar Classes" on page 56
Changes in Version 4 of the IBM 2D Graphics Classes

"Changes in Version 4 of the IBM Collection Classes" on page 72
"Deprecated Functions in Version 4 of the IBM Open Class" on page 73

Changes in Version 4 of the IBM 2D Graphics Classes

This section describes how to migrate your VisualAge C++ 3.0 and 3.5 graphics code to the API available with the IBM Open Class Library. It contains the following:

- General Transitions
- IGraphicContext Replacements
- IGraphicContext Constructors
- IGraphicContext Boundary Accumulation
- IGraphicContext Conversions
- Default Drawing Attributes
- IGraphicContext Device Space Transformation
- IGraphicContext Drawing
- IGraphicContext Drawing Attributes
- Font Operations
- Hit Testing
- IGraphicContext Mapping Modes
- IGraphicContext Recoordination
- IGraphicContext World Space Transformation
- IGraphic
- IGraphic Constructors
- Bounding Rectangle
- Graphic Drawing
- Graphic Bundles
- IGraphic Hit Testing
- IGraphic Object Identifier
- IGraphic World Space Transformation
- IGraphicBundle Constructors

General Transitions

VisualAge C++ 3.0/3.5	IBM Open Class
IGraphicBundle	IGrafBundle
IRegionHandle	no change
ITransformMatrix	IGrafMatrix
IFont	no change
IFont::FaceNameCursor	no change
IFont::PointSizeCursor	no change
IGList::Cursor	IGraphicGroupIterator
IGraphic	IMGraphic
IG3PointArc	IGArcThrough3Points, ICurve

IGArc	IGCurve2D (const IGEllipse2D& e, GDegrees angle1, GDegrees angle2), ICurve
IGBitmap	IGImage, IImage (see ibaseactl/inotebk.cpp)
IGEllipse	IGEllipse2D, IEllipse
IGLine	IGLine2D, ILine
IGList	IGraphicGroup
IGPie	IGPie2D, ICurve
IGChord	IGLoop2D, ILoop (arc constructors)
IGPolyline	IGPolyline2D, IPolyline
IGPolygon	IGPolygon2D, IPolygon
IGRectangle	IGRect2D, IPolygon
IGRegion	IGArea, IArea
IGString	IGraphicText (from Text Framework)
IGraphicContext	IGrafPort

IGraphicContext Replacements

The IGraphicContext class is replaced by IGrafPort and its subclasses, ILinkedGrafPort, IBaseRootGrafPort, IExtendedRootGrafPort, and IStatefulGrafPort as shown in the table below.

VisualAge C++ 3.0/3.5	**IBM Open Class**
IGraphicContext	IGrafPort (abstract)
	ILinkedGrafPort
	IBaseRootGrafPort
	IExtendedRootGrafPort
	IStatefulGrafPort

IGraphicContext Constructors

VisualAge C++ 3.0/3.5	**IBM Open Class**
IGraphicContext (const IPresSpaceHandle&)	IBaseRootGrafPort(const IPresSpaceHandle& deviceContextToBeAdopted, IColorMap* colormap = &(IColorMap::defaultColorMap());
	IExtendedRootGrafPort(IPresSpaceHandle deviceContextToBeAdopted, ICoordinateSystem::EOrientation, EDeviceMappingMode = kPixel);
IGraphicContext() draws to memory	IBaseRootGrafPort() draws on desktop device

IGraphicContext (const IWindowHandle&)	IBaseRootGrafPort(IManagedPresSpaceHandle(const IWindowHandle& windowHandle));
	IExtendedRootGrafPort(IManagedPresSpaceHandle(const IWindowHandle& windowHandle), ICoordinateSystem::EOrientation, EDeviceMappingMode = kPixel);
IGraphicContext(const ISize&)	IGImagePixelAccessor(const IGImage& referencedImage);

IGraphicContext Boundary Accumulation
Area geometries provide a rich set of functions and superior boundary accumulation. For example:

```
IGArea boundary = IGArea(IGRect2D) + IGArea(IGRect2D)
```

Boundary accumulation is done in a device-independent fashion. Instead of using an IGrafPort, call bounds() on a geometry class (such as IGCurve2D), or call geometricBounds on any IMGraphic subclass. Then use the extendTo() function on IGRect2D to accumulate the bounds.

If device-dependent bounds are needed, then call looseFitBounds on IMGraphic, and pass in the appropriate IGrafPort.

VisualAge C++ 3.0/3.5	IBM Open Class
addToBoundingRect	See IGRect2D::extendTo(const IGRect2D&); with geometricBounds() on IMGraphic or bounds() from any geometry class Also see IMGraphic::looseFitBounds (const IGrafPort*) const;
isAccumulatingBoundingRect	
startBoundaryAccumulation	
stopBoundaryAccumulation	

IGraphicContext Conversions

VisualAge C++ 3.0/3.5	IBM Open Class
handle -> IPresSpaceHandle	See IGrafPort::orphanDeviceContext; -> IPresSpaceHandle

IGraphic Current Drawing Position
All IMGraphic classes offer the translateBy(const IGPoint2D&) function for setting the location. In addition, the current drawing position can be set absolutely on IGraphicText, a text subclass of IMGraphic.

VisualAge C++ 3.0/3.5	IBM Open Class
currentDrawingPosition	IGraphicText::location(const IGPoint2D&);

setCurrentDrawingPosition	IGraphicText::setLocation(const IGPoint2D&);
	IMGraphic::translateBy(const IGPoint2D&);

Default Drawing Attributes
Each IGrafPort subclass can maintain state for retained mode drawing. In particular, IBaseRootGrafPort has a set of default values for its attributes.

IGraphicContext Device Space Transformation
For device space transformations, use IGrafMatrix with ILinkedGrafPort.

VisualAge C++ 3.0/3.5	IBM Open Class
pageSize	See IGrafMatrix and ILinkedGrafPort
setPageSize	
setViewPortRect	
viewPortRect	See IGrafPort::invertedDeviceTransform

IGraphicContext Drawing
The draw calls on IGrafPort take the new geometry classes, as well as other information, such as the IAttributeState and the IGrafMatrix. Text is drawn from IGraphicText.

VisualAge C++ 3.0/3.5	IBM Open Class
draw(const IGLine& line)	draw(const IGLine2D& geometry)
draw(const IGPolyline& polyline)	draw(const IGPolyline2D& geometry)
draw(const IGPolygon& polygon)	draw(const IGPolygon2D& geometry)
draw(const IGEllipse& geometry)	draw(const IGEllipse2D& geometry)
draw(const IGRectangle& graphicRectangle)	draw(const IGRect2D& geometry)
draw(const IGArc& arc)	draw(const IGCurve2D& geometry)
draw(const IG3PointArc& arc)	as above, but with IGArcThrough3Points
draw(const IGPie& pie)	draw(const IGLoop2D& geometry) with IGPie2D
draw(const IGChord& chord)	draw(const IGLoop2D& geometry)
draw(const IGString& graphicString)	See IGraphicText
draw(const IGList& list)	See IGraphicGroup: draw(IGrafPort&) const
draw(const IGRegion& region)	draw(const IGArea& geometry)

IGraphicContext Drawing Attributes
IGraphicContext drawing attributes are covered by the IAttribute state and the following subclasses:

- IFillBundle contains imaging information for 2D graphic objects of the solid-filled, no-frame variety.
- IFrameBundle contains imaging information for 2D graphic objects of the non-filled, framed (with any size frame) variety.
- IFillAndFrameBundle contains imaging information for 2D graphic objects of the solid-filled, framed (with any size of frame) variety.

- IGrafBundle is a collection of attributes that contain imaging information for the rendering device.

Unless otherwise stated, all members in the right column are in IAttributeState. Objects of this class are passed to the draw() call on IGrafPort subclasses. Other concepts from IGraphicContext are now expressed as classes such as IPaint, ICap, IJoint, and IPen. IGrafBundle, for example, has setFramePen(const IPen& pen) and const IPen* framePen() const members for relating an IPen.

VisualAge C++ 3.0/3.5	**IBM Open Class**
backgroundColor	none
backgroundMixMode	always transparent
drawOperation	EDrawOperation drawingOperation() const;
fillColor	const IBaseColor* fillColor() const;
fillPattern	const IPaint* fillPaint() const; then call pattern() on IPaint
graphicBundle	See IGrafPort: const IAttributeState* attributes()
mixMode	const IColorTransferMode* fillTransferMode() const;
	const IColorTransferMode* frameTransferMode() const;
	const IImageTransferMode* imageTransferMode() const;
patternOrigin	See IPaint: const IGPoint2D* patternPhase() const;
penColor	const IBaseColor* frameColor() const;
penEndingStyle	const ICap* frameEndCap() const;
penJoiningStyle	const IJoint* frameJoint() const;
penPattern	const IPaint* framePaint() const; then call pattern() on IPaint
penType	EPenType penType() const;
penWidth	GCoordinate penWidth() const;
setBackgroundColor	none
setBackgroundMixMode	always transparent
setDrawOperation	setDrawingOperation(IAttributeState::EDrawOperation);
setFillColor	setFillColor(const IBaseColor&);
setFillPattern	Set the pattern on an IPaint object, then setFillPaint(const IPaint& paint);
setGraphicBundle	See constructors of IGrafPort subclasses
setMixMode	setFillTransferMode(const IColorTransferMode&);
	setFrameTransferMode(const IColorTransferMode&);
	setImageTransferMode(const IImageTransferMode&);

setPatternOrigin	See IPaint (patternPhase means origin):
	IPaint(const IBaseColor& aColor, const IMaskPattern& maskPattern = IMaskPattern::solid(), const IGPoint2D& patternPhase = IGPoint2D::origin());
	IPaint(const IGImage& imagePattern, const IGPoint2D& patternPhase = IGPoint2D::origin());
setPenColor	setFrameColor(const IBaseColor& color);
setPenEndingStyle	setFrameEndCap(const ICap&);
setPenJoiningStyle	setFrameJoint(const IJoint&);
setPenPattern	Set pattern on an IPaint object, then setFramePaint(const IPaint& paint);
setPenType	Set pen type on an IPen object, then setFramePen(const IPen& newPen);
setPenWidth	Set pen width on an IPen object, then setFramePen(const IPen& newPen);

Font Operations
Use ITextStyleSet instead of IFont when setting font attributes such as bold. Use IFont only for font manager purposes. See Styles and Style Sets for more information.

Hit Testing
The API currently provides simple support for hit testing, which does not replace the full functionality originally provided in VisualAge 3.5. A simple approach is to use the geometricBounds call to obtain an IGRect2D, and then to ask that rectangle if it contains a point or intersects with a rectangle.

From IMGraphic:

```
IGRect2D geometricBounds() const;
```

From IGRect2D:

```
bool contains(const IGPoint2D&) const;

bool intersects(const IGRect2D&) const;
```

IGraphicContext Mapping Modes
IExtendedRootGrafPort is an extended IBaseRootGrafPort. It defines the coordinate system and mapping mode of the device.

VisualAge C++ 3.0/3.5	IBM Open Class
mappingMode	Set at construction time only
setMappingMode	IExtendedRootGrafPort(IPresSpaceHandle deviceContextToBeAdopted, ICoordinateSystem::EOrientation, EDeviceMappingMode = kPixel);

IGraphicContext Recoordination
The recoordination height, which is used to convert to and from coordinates with the origin in upper left and lower left, is set by IExtendedRootGrafPort at the time of construction. Apply an ILinkedGrafPort with an IGrafMatrix of type

EMatrixKind::kModelMatrix to change it.

VisualAge C++ 3.0/3.5	IBM Open Class
recoordinationHeight	Retrieve from the model matrix
setRecoordinationHeight	IExtendedRootGrafPort and ILinkedGrafPort

IGraphicContext Region Operations
All IGrafPort subclasses can return the clip area. Use ILinkedGrafPort with an IGArea for clipping. Additionally, IStatefulGrafPort can set it directly.

VisualAge C++ 3.0/3.5	IBM Open Class
setClipRegion	ILinkedGrafPort(IGrafPort* referencedParent, const IGArea* referencedClipArea);
	See IStatefulGrafPort: setClipArea(const IGArea& clipArea);
clipRegion	SeeIGrafPort: const IGArea* clipArea() const;
clearClipRegion	set a huge area

IGraphicContext World Space Transformation
Use ILinkedGrafPort with an IGrafMatrix for a world space transform. Specify EMatrixKind::kModelMatrix to transform just the geometry, or select EMatrixKind::kViewMatrix to transform both the geometry and the attributes (for example, pen width).

VisualAge C++ 3.0/3.5	IBM Open Class
setWorldTransformMatrix	ILinkedGrafPort(IGrafPort* referencedParent, EMatrixKind matrixKind, const IGrafMatrix* referencedMatrix);
	setMatrix(EMatrixKind matrixKind, const IGrafMatrix& matrix);
worldTransformMatrix	const IGrafMatrix* matrix(EMatrixKind) const;

IGraphic
Just as an IGraphic can contain an IGraphicBundle and an ITransformationMatrix, the new class IMGraphic can contain an IAttributeState and an IGrafMatrix. Subclasses of IGraphic have to override drawOn, whereas those of IMGraphic must override draw() as well as geometricBounds and transformBy.

IGraphic Constructors
Both IGraphic and IMGraphic are abstract base classes, and as such have no public constructors. As exhibited by the IGraphic hierarchy, the IMGraphic tree follows the pattern of providing default, copy, and bundle constructors.

VisualAge C++ 3.0/3.5	IBM Open Class	
	Model	Geometry
default - uses attributes in drawOn	same, but for draw	see IGrafPort::draw

copy - uses copy's attributes	same	ILinkedGrafPort(IGrafPort* referencedParent, const IAttributeState* referencedAttributes);
IGraphicBundle	IAttributeState	

Bounding Rectangle
The bounding rectangle concept is similar. Call looseFitBounds for more accurate bounds at device resolution and with respect to the attributes.

VisualAge C++ 3.0/3.5	IBM Open Class
boundingRect	IGRect2D geometricBounds() const
	IGRect2D looseFitBounds() const

Graphic Drawing
IGrafPort replaces IGraphicContext, and draw(IGrafPort&) replaces drawOn(IGraphicContext&).

VisualAge C++ 3.0/3.5	IBM Open Class
drawOn	draw(IGrafPort&) const

Graphic Bundles
IGrafBundle replaces IGraphicBundle.

VisualAge C++ 3.0/3.5	IBM Open Class
graphicBundle	const IGrafBundle* bundle( ) const;
hasGraphicBundle	check if the bundle function returns a null pointer
removeGraphicBundle	IGrafBundle* orphanBundle();
setGraphicBundle	adoptBundle(IGrafBundle* adoptedBundle);

IGraphic Hit Testing
Full support for hit detection is not currently provided. For more information see the section on Hit Testing on this page.

IGraphic Object Identifier
There is no corresponding identification mechanism in the IBM Open Class. You can use an IRawArray or a dictionary-like collection to maintain keys.

IGraphic World Space Transformation
An IGraphic class supports replacing, pre-multiplying, and post-multiplying transformations. For IMGraphic, transformation is always post-multiply and cannot be undone, because the associated geometry has been transformed. To transform the view but retain the geometry, use EMatrixKind::kViewMatrix with ILinkedGrafPort instead.

In IGraphic classes, setTransformMatrix replaces the current transform, if any. IMGraphic classes, however, concatenate the transform in transformBy, and do not necessarily store the matrix. This results in a lighter-weight, high-performance object

VisualAge C++ 3.0/3.5	IBM Open Class
ITransformMatrix	In IGraphicGroup only: const IGrafMatrix* matrix() const;
setTransformMatrix	transformBy(const IGrafMatrix&)
hasTransformMatrix	In IGraphicGroup only: test NIL != const IGrafMatrix* matrix() const;
resetTransformMatrix	use the inverse of the matrix
rotateBy	rotateBy(GDegrees, const IGPoint2D& centerOfRotation= IGPoint2D::origin());
scaleBy	scaleBy(const IGPoint2D&, const IGPoint2D& centerOfScale= IGPoint2D::origin());
translateBy	translateBy(const IGPoint2D&);
setTransformMethod	always post-multiply
transformMethod	always post-multiply

IGraphicBundle Constructors

The class IAttributeState resembles IGraphicBundle's functionality with the subclasses IFillBundle and IFrameBundle. IFillAndFrameBundle combines these to replace most of the functions. IGrafBundle subsumes IGraphicBundle and adds more functionality, such as the image patterns available through IPaint.

Constructors are described below, and drawing attributes are addressed in the preceding section on IGraphicContext, which uses IGraphicBundle.

VisualAge C++ 3.0/3.5	IBM Open Class
default	IGrafBundle() - no attributes set
copy	IGrafBundle(const IGrafBundle&); copies attributes
from IGraphicContext	IGrafBundle(const IAttributeState&);
	IGrafBundle(const IPaint& fPaint, IAttributeState::EDrawOperation attribute = kFrame);
	IGrafBundle(const IPaint& fillPaint, const IPaint& framePaint, IAttributeState::EDrawOperation attribute = kFillAndFrame);
	IGrafBundle(const IBaseColor& color, IAttributeState::EDrawOperation attribute = kFrame);
	IGrafBundle(const IBaseColor& fillColor, const IBaseColor& frameColor, IAttributeState::EDrawOperation attribute = kFillAndFrame);

RELATED CONCEPTS

Changes in Version 4 of the IBM Collection Classes

Backward-Compatible Items
The following items from former releases are compatible with this release:

- **Reference Classes**
  Reference classes are no longer necessary for polymorphic use of the collections. The concrete collection classes are now directly derived from the abstract class hierarchy. A linkage of abstract and concrete classes through reference classes is therefore superfluous. Nevertheless you can continue using the reference class syntax in existing programs.
- **IIterator and IConstantIterator**
  The classes IIterator and IConstantIterator are now called IApplicator and IConstantApplicator. The new names express more precisely what the purpose of objects from these classes is: They do not iterate over a collection themselves, but they provide a function that is applied to the elements of a collection during iteration with allElementsDo.

  The classes IIterator and IConstantIterator are still available but not recommended.
- **The forCursor macro**
  Instead of the forCursor macro, the forICursor macro is introduced. The forCursor macro is still available. However, as with the iterator classes, it is recommended that you use the new version.
- **IECOps**
  Previously all implementation variants of the collections bag, set, sorted bag and sorted set used the element operation class IECOps. Now these collections require only class ICOps which is a subset of IECOps. IECOps is no longer needed, yet it is still available.
- **Naming Conventions**
  New names have been introduced for the implementation variants as well as for the corresponding header files. The old names can still be used in existing programs. Consider the key set as example:

Old Names		**New Names**	
IKeySet	ikeyset.h	IKeySet	iks.h
		IKeySetAsAvlTree	iksavl.h
IKeySetOnBSTKeySortedSet	iksbst.h	IKeySetAsBstTree	iksbst.h
IHashKeySet	ihshks.h	IKeySetAsHshTable	ikshsh.h
IKeySetOnSortedLinkedSequence	ikssls.h	IKeySetAsList	ikslst.h
IKeySetOnSortedTabularSequence	iksstab.h	IKeySetAsTable	ikstab.h
IKeySetOnSortedDilutedSequence	iksdil.h	IKeySetAsDilTable	iksdil.h

Incompatibilities
The following items are not compatible with the new collection class library release:

- **New class hierarchy**
  The abstract hierarchy makes use of virtual inheritance. When you derive from a Collection Class and implement your own copy constructor, you must initialize the virtual base class IACollection<Element> in your derived classes.
- **The newCursor method**
  In contrast to previous releases, the return type of the newCursor method is now for any collection a pointer to the abstract cursor class ICursor (ICursor*).
- **Deriving from Reference Classes**
  You can still derive from reference classes without overriding existing collection class member functions. Yet, you can no longer override existing collection class functions and use your derived collection class in a polymorphic way without additional effort.

RELATED CONCEPTS

"Chapter 3. Changes in Version 4 of IBM Open Class" on page 41
"AIX Changes in Version 4 of the IBM Open Class" on page 44
"Changes in Version 4 of the IWindow Class" on page 46
"Changes in Version 4 of the Handler Classes" on page 50
"Changes in Version 4 of the Canvas Classes" on page 53
"Changes in Version 4 of the Toolbar Classes" on page 56
"**Other Changes in Version 4 of the IBM User Interface Classes**" on page 58
Changes in Version 4 of the IBM 2D Graphics Classes
"Deprecated Functions in Version 4 of the IBM Open Class"

Deprecated Functions in Version 4 of the IBM Open Class

As the Open Class Library functionality increases, the interface must be changed to improve the quality and design. Deprecated interfaces are listed below so you can migrate to replacement classes and functions.

There are several important guidelines regarding deprecated functions:

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

The following are changes in the language, functions, enums, and types in Version 4 of the IBM Open Class Library:

Type	Name Before Version 4	Name in Version 4 and Later
class	IBase	
class	IBase ::Version	IVersion

Type	Name Before Version 4	Name in Version 4 and Later
function	IBase ::asString() const	IStringGenerator ::stringFor() const (where the IStringGenerator was constructed passing in an instance of an IStringGeneratorasString generator function)
class	IBasicColor	IBaseColor
class	IGArc	IGCurve2D
class	IGBitmap	IGImage, IImage
class	IGEllipse	IGEllipse2D, IEllipse
class	IGLine	IGLine2D, ILine
class	IGList	IGraphicGroup
class	IGPie	IGPie2D, ICurve
class	IGPolyline	IGPolyline2D, IGPolyline
class	IGraphic	IMGraphic
class	IGraphicBundle	IGrafBundle
class	IGraphicContext	IGrafPort
class	IGRectangle	IGRect2D, IPolygon
class	IGRegion	IGArea, IArea
class	IGString	IGraphicText
class	IRootGrafPort	IExtendedRootGrafPort
class	ITransformMatrix	
class	IWindow ::BidiSettings	IBidiSettings
class	IMouseConnectionTo	IMouseClickConnectionTo and IMouseMoveConnectionTo
typedef	Boolean	bool
typedef	IBaseErrorInfo ::IErrorInfo	IBaseErrorInfo
typedef	IColor ::SysColor	IColor ::SystemColor
typedef	IContextHandle	IAnchorBlockHandle
typedef	ICoordinateSystem ::Orientation	ICoordinateSystem ::EOrientation
enum	IColor ::Color	IBaseColor ::EPredefinedColor
enum	IColor ::SystemColor	IColor ::ESystemColor
enum	EDeviceCoordinateSystem	ICoordinateSystem ::EOrientation
enum	IImage ::EDitherType	IGImagePixelAccessor ::EDitherType

Type	Name Before Version 4	Name in Version 4 and Later
enum	IImage ::EImageFormat	IGImagePixelAccessor ::EImageFormat
enum	INonGUIApplication ::PriorityClass	INonGUIApplication ::EProcessPriority
enum	IWindow ::BidiLayout	IBidiSettings ::BidiLayout
enum	IWindow ::BidiNumeralType	IBidiSettings ::BidiNumeralType
enum	IWindow ::BidiTextOrientation	IBidiSettings ::BidiTextOrientation
enum	IWindow ::BidiTextShape	IBidiSettings ::BidiTextShape
enum	IWindow ::BidiTextType	IBidiSettings ::BidiTextType
constructor	IBidiSettings (const IGraphicContext&)	IBidiSettings(const IGrafPort&)
constructor	IBitFlag (unsigned long)	IBitFlag(unsigned long, const unsigned long[])
constructor	IBitFlag (unsigned long, unsigned long)	IBitFlag(unsigned long, const unsigned long[])
constructor	IColor(Color color)	IColor(IColor ::EPredefinedColor)
constructor	IColor(SystemColor value)	IColor(IColor ::ESystemColor)
constructor	IColor(unsigned long pixel)	IColor(long index)
constructor	IImage(IPresSpaceHandle psh, int , int)	
constructor	IImage(const IString& .....)	IGImagePixelAccessor
data	IEntryField ::lowerCase	Create a handler to convert characters to lowercase
data	IEntryField ::upperCase	Create a handler to convert characters to uppercase
data	IKeyboardEvent ::ulAltFlag	
data	IKeyboardEvent ::ulAltMask	
data	IKeyboardEvent ::ulCharacterFlag	
data	IKeyboardEvent ::ulCompositeFlag	
data	IKeyboardEvent ::ulCtrlFlag	
data	IKeyboardEvent ::ulCtrlMask	
data	IKeyboardEvent ::ulForCompositeFlag	

Type	Name Before Version 4	Name in Version 4 and Later
data	IKeyboardEvent ::ulInvalidCompositeFlag	
data	IKeyboardEvent ::ulRepeatedFlag	
data	IKeyboardEvent ::ulScanCodeFlag	
data	IKeyboardEvent ::ulShiftFlag	
data	IKeyboardEvent ::ulShiftMask	
data	IKeyboardEvent ::ulUncombinedFlag	
data	IKeyboardEvent ::ulUpTransitionFlag	
data	IKeyboardEvent ::ulVirtualFlag	
constructor	INonGUIThread(const IReference<IThreadFn>&)	INonGUIThread(IThreadFn*)
function	IBidiSettings ::apply(const IGraphicContext&)	apply(const IGrafPort&)
function	IBitFlag ::asExtendedUnsignedLong ( ) const	No true replacement. Classes using IBitFlag objects must provide their own mapping logic (IWindow ::convertToGUIStyle is an example).
function	IBitFlag ::asUnsignedLong ( ) const	No true replacement. Classes using IBitFlag objects must provide their own mapping logic (IWindow ::convertToGUIStyle is an example)
function	IBitFlag ::setValue(unsigned long, unsigned long)	IBitFlag ::setValue(const IBitFlag&)
function	ICanvas ::defaultPushButton ( ) const	IWindow ::defaultEmphasisButton
function	ICanvas ::origDefaultButtonHandle ( ) const	IWindow ::defaultPushButton
function	IClipboard ::setOwner	Specify the owner when constructing the IClipboard (construct a new IClipboard object if necessary)
function	IColor ::asPixel() const	IColor ::index()
function	IColor ::systemColor() const	IColor ::runtimeGuiColor()

Type	Name Before Version 4	Name in Version 4 and Later
function	IColor ::value() const	IColor ::predefinedColor
function	ICurrentThread ::appContext	ICurrentThread ::anchorBlock
function	ICurrentThread ::appShell	IThread ::applicationShell or IWindow ::desktopWindow()->handle()
function	ICustomButtonDrawEvent ::graphicContext	ICustomButtonDrawEvent ::grafPort
function	IDrawingCanvas ::graphicContext	IDrawingCanvas ::grafPort
function	IDrawingCanvas ::graphicList	IDrawingCanvas ::graphicGroup
function	IDrawingCanvas ::setGraphicContext	IDrawingCanvas ::setGrafPort
function	IDrawingCanvas ::setGraphicList	IDrawingCanvas ::setGraphicGroup
function	IEntryField ::hasChanged	IEntryField ::hasTextChanged
function	IEntryField ::setChangedFlag	IEntryField ::setTextChangedFlag
function	IImage ::loadFromFile	IGImagePixelAccessor ::loadFromFile
function	IImage ::writeToFile	IGImagePixelAccessor ::writeToFile
function	IInfoArea ::handleEventsFor	IInfoArea ::startShowingMenuInfoFor
function	IInfoArea ::inactiveText	disabledText
function	IInfoArea ::setInactiveText	setDisabledText
function	IInfoArea ::stopHandlingEventsFor	IInfoArea ::stopShowingMenuInfoFor
function	IMultiLineEdit ::setChangedFlag	IMultiLineEdit ::setTextChangedFlag
function	IMultiLineEdit ::resetChangedFlag	IMultiLineEdit ::resetTextChangedFlag
function	IMultiLineEdit ::isChanged	IMultiLineEdit ::hasTextChanged
function	INonGUIApplication ::adjustPriority	INonGUIApplication ::setProcessPriority
function	INonGUIApplication ::setPriority	INonGUIApplication ::setProcessPriority
function	INonGUIThread ::priorityLevel	INonGUIThread ::threadPriority
function	INonGUIThread ::setPriority	INonGUIThread ::setThreadPriority

Type	Name Before Version 4	Name in Version 4 and Later
function	INonGUIThread ::~	
function	INotificationEvent ::setNotifierAttrChanged (booltrue)	
function	INotificationEvent ::setEventData(const IEventData&)	
function	INotificationEvent ::setObserverData(const IEventData&)	
function	INotificationEvent ::hasNotifierAttrChanged	
function	INotificationEvent ::eventData	
function	INotificationEvent ::observerData	
function	INotificationEvent ::notifier	
function	INotifier ::addObserver(IObserver&, const IEventData&)	
function	IPaintEvent ::setGraphicContext	IPaintEvent ::setGrafPort
function	IResourceLibrary ::loadAccelTable(unsigned long, const IWindowHandle&)	IResourceLibrary ::loadAccelTable(unsigned long)
function	ISetCanvas ::setText	ICanvas ::setBorderText
function	ISetCanvas ::text	ICanvas ::borderText
function	IStandardNotifier ::addObserver(IObserver&, const IEventData&)	
function	IWindow ::addObserver(IObserver&, const IEventData&)	
function	IWindow ::applyBidiSettings	IBidiSettings ::apply
function	IWindow ::enableFastWindowWithHandle (bool enable = true)	IWindow ::reserveUserWindowWord
function	IWindow ::isBidiSupported	IBidiSettings ::isBidiSupported
function	IWindow:: isFastWindowWithHandleEnabled()	IWindow:: isUserWindowWordReserved
constant	True	true

Type	Name Before Version 4	Name in Version 4 and Later
constant	False	false
constant	ICommand ::kCopyToId	kSaveAsId
constant	ICoordinateSystem ::originLowerLeft	ICoordinateSystem ::kOriginLowerLeft
constant	ICoordinateSystem ::originUpperLeft	ICoordinateSystem ::kOriginUpperLeft
constant	IHelpWindow ::using	usingHelp
constant	IKeyboardEvent ::ulAltFlag	
constant	IKeyboardEvent ::ulAltMask	isAltDown()
constant	IKeyboardEvent ::ulCharacterFlag	
constant	IKeyboardEvent ::ulCompositeFlag	
constant	IKeyboardEvent ::ulCtrlFlag	
constant	IKeyboardEvent ::ulCtrlMask	isCtrlDown()
constant	IKeyboardEvent ::ulForCompositeFlag	
constant	IKeyboardEvent ::ulInvalidCompositeFlag	
constant	IKeyboardEvent ::ulRepeatFlag	
constant	IKeyboardEvent ::ulScanCodeFlag	
constant	IKeyboardEvent ::ulShiftFlag	
constant	IKeyboardEvent ::ulShiftMask	isShiftDown()
constant	IKeyboardEvent ::ulUncombinedFlag	
constant	IKeyboardEvent ::ulUpTransitionFlag	
constant	IKeyboardEvent ::ulVirtualFlag	
constant	IProgressIndicator ::armTickOffsetId	armChangeId
constant	IProgressIndicator ::armPixelOffsetId	armChangeId
constant	IScrollBar ::autoSize	
constant	IScrollBar ::scrollBoxPositionId	scrollBoxChangeId, scrollBoxTrackId

Type	Name Before Version 4	Name in Version 4 and Later
constant	IString ::and	bitAnd
constant	IString ::or	bitOr
constant	IString ::exclusiveOr	bitExclusiveOr
constant	IToolBar ::hidden	
constant	IToolBar ::noDragDrop	
constant	IToolBarButton ::noDragDelete	
constant	kLeftHand	ICoordinateSystem ::kOriginUpperLeft
constant	kRightHand	ICoordinateSystem ::kOriginLowerLeft

RELATED CONCEPTS

"Chapter 3. Changes in Version 4 of IBM Open Class" on page 41
"AIX Changes in Version 4 of the IBM Open Class" on page 44
"Changes in Version 4 of the IWindow Class" on page 46
"Changes in Version 4 of the Handler Classes" on page 50
"Changes in Version 4 of the Canvas Classes" on page 53
"Changes in Version 4 of the Toolbar Classes" on page 56
"**Other Changes in Version 4 of the IBM User Interface Classes**" on page 58
Changes in Version 4 of the IBM 2D Graphics Classes
"Changes in Version 4 of the IBM Collection Classes" on page 72