CICS Transaction Server for z/OS
5.4

C++ OO Class Libraries

Note

Before using this information and the product it supports, read the information in "Notices" on page 261.

This edition applies to the IBM CICS® Transaction Server for z/OS® Version 5 Release 4 (product number 5655-Y04) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1974, 2023.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this PDF ix

Chapter 1. Installation and setup 1
- Getting ready for object oriented CICS 1
- Installed contents 1
  - Header files 1
  - Dynamic link library 2
  - Sample source code 2
  - Other data sets for CICS Transaction Server for z/OS 3

Chapter 2. Using the CICS foundation classes 5
- C++ Objects 5
  - Creating an object 5
  - Using an object 6
  - Deleting an object 6
- Overview of the foundation classes 6
  - Base classes 6
  - Resource identification classes 7
  - Resource classes 8
  - Support Classes 10
  - Using CICS resources 11
- Buffer objects 12
  - IccBuf class 12
- Using CICS Services 15
  - File control 15
  - Program control 19
  - Starting transactions asynchronously 20
  - Transient Data 23
  - Temporary storage 25
  - Terminal control 27
  - Time and date services 28
- Compiling, executing, and debugging 30
  - Compiling Programs 30
  - Executing Programs 31
  - Program debugging 31
- Conditions, errors, and exceptions 32
  - Foundation Class Abend codes 32
  - C++ Exceptions and the Foundation Classes 32
  - CICS conditions 34
  - Platform differences 36
- Polymorphic Behavior 38
  - Example of polymorphic behavior 39
- Storage management 40
- Parameter passing conventions 41
- Scope of data in IccBuf reference returned from 'read' methods 41

Chapter 3. Foundation Classes: reference 43
- Mapping EXEC CICS calls to Foundation Class methods 44
- Mapping Foundation Class methods to EXEC CICS calls 49
- Icc structure 56

Functions ... 56
Enumerations ... 58
IccAbendData class ... 60
IccAbendData constructor (protected) ... 60
Public methods ... 60
Inherited public methods ... 64
Inherited protected methods ... 65
IccAbsTime class ... 65
IccAbsTime constructor ... 65
Public methods ... 66
Inherited public methods ... 69
Inherited protected methods ... 69
IccAlarmRequestId class ... 70
IccAlarmRequestId constructors ... 70
Public methods ... 71
Inherited public methods ... 72
Inherited protected methods ... 72
IccBase class ... 72
IccBase constructor (protected) ... 72
Public methods ... 73
Protected methods ... 74
Enumerations ... 75
IccBuf class ... 76
IccBuf constructors ... 76
Public methods ... 78
Inherited public methods ... 86
Inherited protected methods ... 86
Enumerations ... 86
IccClock class ... 87
IccClock constructor ... 87
Public methods ... 87
Inherited public methods ... 90
Inherited protected methods ... 91
Enumerations ... 91
IccCondition structure ... 92
Enumerations ... 92
IccConsole class ... 93
IccConsole constructor (protected) ... 94
Public methods ... 94
Inherited public methods ... 96
Inherited protected methods ... 97
Enumerations ... 97
IccControl class ... 97
IccControl constructor (protected) ... 97
Public methods ... 98
Inherited public methods ... 101
Inherited protected methods ... 102
IccConvId class ... 102
IccConvId constructors ... 102
Public methods ... 103
Inherited public methods ... 103
Inherited protected methods ... 103
IccDataQueue class ... 104
IccDataQueue constructors ... 104
Public methods ... 104
Inherited public methods ... 106
Inherited protected methods ... 106
IccDataQueueId class ... 107

IccDataQueueId constructors ... 107
Public methods ... 107
Inherited public methods ... 108
Inherited protected methods ... 108
IccEvent class ... 108
IccEvent constructor ... 108
Public methods ... 109
Inherited public methods ... 110
Inherited protected methods ... 110
IccException class ... 110
IccException constructor ... 110
Public methods ... 111
Inherited public methods ... 113
Inherited protected methods ... 113
Enumerations ... 113
IccFile class ... 114
IccFile constructors ... 114
Public methods ... 115
Inherited public methods ... 124
Inherited protected methods ... 125
Enumerations ... 125
IccFileId class ... 126
IccFileId constructors ... 126
Public methods ... 127
Inherited public methods ... 127
Inherited protected methods ... 128
IccFileIterator class ... 128
IccFileIterator constructor ... 128
Public methods ... 129
Inherited public methods ... 130
Inherited protected methods ... 130
IccGroupId class ... 131
IccGroupId constructors ... 131
Public methods ... 131
Inherited public methods ... 132
Inherited protected methods ... 132
IccJournal class ... 132
IccJournal constructors ... 132
Public methods ... 133
Inherited public methods ... 136
Inherited protected methods ... 136
Enumerations ... 136
IccJournalId class ... 137
IccJournalId constructors ... 137
Public methods ... 138
Inherited public methods ... 138
Inherited protected methods ... 139
IccJournalTypeId class ... 139
IccJournalTypeId constructors ... 139
Public methods ... 139
Inherited public methods ... 140
Inherited protected methods ... 140
IccKey class ... 140
IccKey constructors ... 141
Public methods ... 141
Inherited public methods ... 144
Inherited protected methods ... 144
Enumerations ... 144

IccLockId class ... 144
IccLockId constructors ... 144
Public methods ... 145
Inherited public methods ... 145
Inherited protected methods ... 146
IccMessage class ... 146
IccMessage constructor ... 146
Public methods ... 147
Inherited public methods ... 147
Inherited protected methods ... 148
IccPartnerId class ... 148
IccPartnerId constructors ... 148
Public methods ... 148
Inherited public methods ... 149
Inherited protected methods ... 149
IccProgram class ... 150
IccProgram constructors ... 150
Public methods ... 150
Inherited public methods ... 153
Inherited protected methods ... 153
Enumerations ... 153
IccProgramId class ... 154
IccProgramId constructors ... 154
Public methods ... 154
Inherited public methods ... 155
Inherited protected methods ... 155
IccRBA class ... 155
IccRBA constructor ... 155
Public methods ... 156
Inherited public methods ... 157
Inherited protected methods ... 157
IccRecordIndex class ... 158
IccRecordIndex constructor (protected) ... 158
Public methods ... 158
Inherited public methods ... 159
Inherited protected methods ... 159
Enumerations ... 159
IccRequestId class ... 159
IccRequestId constructors ... 159
Public methods ... 160
Inherited public methods ... 161
Inherited protected methods ... 161
IccResource class ... 161
IccResource constructor (protected) ... 161
Public methods ... 162
Inherited public methods ... 167
Inherited protected methods ... 167
Enumerations ... 167
IccResourceId class ... 168
IccResourceId constructors (protected) ... 168
Public methods ... 168
Protected methods ... 169
Inherited public methods ... 169
Inherited protected methods ... 169
IccRRN class ... 169
IccRRN constructors ... 170
Public methods ... 170
Inherited public methods ... 171

Inherited protected methods... 171
IccSemaphore class... 172
IccSemaphore constructor... 172
Public methods... 173
Inherited public methods... 174
Inherited protected methods... 174
Enumerations... 174
IccSession class... 175
IccSession constructors (public)... 175
IccSession constructor (protected)... 176
Public methods... 176
Inherited public methods... 184
Inherited protected methods... 185
Enumerations... 185
IccStartRequestQ class... 186
IccStartRequestQ constructor (protected)... 186
Public methods... 186
Inherited public methods... 191
Inherited protected methods... 192
Enumerations... 192
IccSysId class... 192
IccSysId constructors... 192
Public methods... 193
Inherited public methods... 193
Inherited protected methods... 193
IccSystem class... 194
IccSystem constructor (protected)... 194
Public methods... 194
Inherited public methods... 198
Inherited protected methods... 199
Enumerations... 199
IccTask class... 200
IccTask Constructor (protected)... 200
Public methods... 200
Inherited public methods... 208
Inherited protected methods... 209
Enumerations... 209
IccTempStore class... 211
IccTempStore constructors... 211
Public methods... 212
Inherited public methods... 214
Inherited protected methods... 215
Enumerations... 215
IccTempStoreId class... 216
IccTempStoreId constructors... 216
Public methods... 216
Inherited public methods... 217
Inherited protected methods... 217
IccTermId class... 217
IccTermId constructors... 217
Public methods... 218
Inherited public methods... 218
Inherited protected methods... 219
IccTerminal class... 219
IccTerminal constructor (protected)... 219
Public methods... 219
Inherited public methods... 233
Inherited protected methods... 233

Enumerations.... 234
IccTerminalData class.... 235
IccTerminalData constructor (protected).... 235
Public methods.... 235
Inherited public methods.... 241
Inherited protected methods.... 241
IccTime class.... 242
IccTime constructor (protected).... 242
Public methods.... 242
Inherited public methods.... 243
Inherited protected methods.... 244
Enumerations.... 244
IccTimeInterval class.... 244
IccTimeInterval constructors.... 244
Public methods.... 245
Inherited public methods.... 245
Inherited protected methods.... 246
IccTimeOfDay class.... 246
IccTimeOfDay constructors.... 246
Public methods.... 247
Inherited public methods.... 248
Inherited protected methods.... 248
IccTPNameId class.... 248
IccTPNameId constructors.... 249
Public methods.... 249
Inherited public methods.... 250
Inherited protected methods.... 250
IccTransId class.... 250
IccTransId constructors.... 250
Public methods.... 251
Inherited public methods.... 251
Inherited protected methods.... 252
IccUser class.... 252
IccUser constructors.... 252
Public methods.... 253
Inherited public methods.... 255
Inherited protected methods.... 255
IccUserId class.... 256
IccUserId constructors.... 256
Public methods.... 256
Inherited public methods.... 257
Inherited protected methods.... 257
IccValue structure.... 257
Enumeration.... 257
main function.... 258

Notices.... 261

Index.... 267

About this PDF

This PDF describes how to use the CICS C++ foundation classes, which allow an application programmer to access CICS services that are available via the EXEC CICS API.

For details of the terms and notation used, see Conventions and terminology used in the CICS documentation in IBM Knowledge Center.

Date of this PDF

This PDF was created on 2024-01-04 (Year-Month-Date).

© Copyright IBM Corp. 1974, 2023

Chapter 1. Installation and setup

This section describes the CICS foundation classes installed on your CICS server.

Getting ready for object oriented CICS

You must be familiar with object oriented concepts and technology, the C++ language and with CICS in order to understand the topics that follow.

This is not intended to be an introduction to any of these subjects.

Installed contents

The CICS foundation classes package consists of several files or data sets.

The CICS foundation classes package consists of several files or data sets. These contain the:

- header files
- executables (DLL's)
- samples
- other CICS Transaction Server for z/OS files

This section describes the files that comprise the CICS C++ Foundation Classes and explains where you can find them on your CICS server.

Header files

The header files are the C++ class definitions needed to compile CICS C++ Foundation Class programs.

C++ Header File	Classes Defined in this Header
ICCABDEH	IccAbendData
ICCBASEH	IccBase
ICCBUFEH	IccBuf
ICCCLKEH	IccClock
ICCCNDEH	IccCondition (struct)
ICCCONEH	IccConsole
ICCCTLEH	IccControl
ICCDATEH	IccDataQueue
ICCEH	see “1” on page 2
ICCEVTEH	IccEvent
ICCEXCEH	IccException
ICCFILEH	IccFile
ICCFLIEH	IccFileIterator
ICCGLBEH	Icc (struct) (global functions)
ICCJRNEH	IccJournal
ICCMSGEH	IccMessage
ICCPRGEH	IccProgram

© Copyright IBM Corp. 1974, 2023

C++ Header File	Classes Defined in this Header
ICCRECEH	IccRecordIndex, IccKey, IccRBA and IccRRN
ICCRESEH	IccResource
ICCRIDEH	IccResourceId + subclasses (such as IccConvId)
ICCSEMEH	IccSemaphore
ICCSESEH	IccSession
ICCSRQEH	IccStartRequestQ
ICCSYSEH	IccSystem
ICCTIMEH	IccTime, IccAbsTime, IccTimeInterval, IccTimeOfDay
ICCTMDEH	IccTerminalData
ICCTMPEH	IccTempStore
ICCTRMEH	IccTerminal
ICCTSKEH	IccTask
ICCUSREH	IccUser
ICCVALEH	IccValue (struct)

Note:

1. A single header that #includes all the listed header files is supplied as `ICCEH`
2. The file `ICCMAIN` is also supplied with the C++ header files. This contains the **main** function stub that should be used when you build a Foundation Class program.
3. Header files are located in CICSTS54.CICS .SDFHC370.

Location

PDS: CICSTS54.CICS.SDFHC370.

Dynamic link library

The Dynamic Link Library is the runtime environment that is needed to support a CICS C++ Foundation Class program.

Location

`ICCFCDLL` module in PDS: CICSTS54.CICS.SDFHLOAD.

Sample source code

The samples are provided to help you understand how to use the classes to build object oriented applications.

Location

PDS: CICSTS54.CICS.SDFHSAMP.

Running the sample applications

If you have installed the resources defined in the member DFHCURDS, you should be ready to run some of the sample applications.

The sample programs are supplied as source code in library CICSTS54.CICS.SDFHSAMP and before you can run the sample programs, you need to compile, pre-link and link them. To do this, use the procedure ICCFCCL in data set CICSTS54.CICS.SDFHPROC.

ICCFCCL contains the Job Control Language needed to compile, pre-link and link a CICS user application. Before using ICCFCCL you may find it necessary to perform some customization to conform to your installation standards. See also Compiling programs.

Sample programs such as ICCBUF, ICCCLK and ICC$HEL require no additional CICS resource definitions, and should now execute successfully.

Other sample programs, in particular the DTP samples named ICC$SES1 and ICC$SES2, require additional CICS resource definitions. Refer to the prologues in the source of the sample programs for information about these additional requirements.

Other data sets for CICS Transaction Server for z/OS

CICSTS54.CICS.SDFHSDCK contains the member

- ICCFCIMP - 'sidedeck' containing import control statements

CICSTS54.CICS.SDFHPROC contains the members

- ICCFCC - JCL to compile a CFC user program
- ICCFCCL - JCL to compile, prelink and link a CFC user program
- ICCFCGL - JCL to compile and link an XPLINK program that uses CFC libraries.
- ICCFCL - JCL to prelink and link a CFC user program

CICSTS54.CICS.SDFHLOAD contains the members

- DFHCURDS - program definitions required for CICS system definition.
- DFHCURDI - program definitions required for CICS system definition.

Chapter 2. Using the CICS foundation classes

This section describes the CICS foundation classes and how to use them. There is a formal listing of the user interface in Foundation Classes: reference.

C++ Objects

This section describes how to create, use, and delete objects.

This section describes how to create, use, and delete objects. In our context an object is an instance of a class. An object cannot be an instance of a base or abstract base class. It is possible to create objects of all the concrete (non-base) classes described in the reference part of this book.

Creating an object

If a class has a constructor it is executed when an object of that class is created. This constructor typically initializes the state of the object. Foundation Classes' constructors often have mandatory positional parameters that the programmer must provide at object creation time.

C++ objects can be created in one of two ways:

1. Automatically, where the object is created on the C++ stack. For example:

   ```
   {
   ClassX objX
   ClassY objY(parameter1);
   } //objects deleted here
   ```

   Here, objX and objY are automatically created on the stack. Their lifetime is limited by the context in which they were created; when they go out of scope they are automatically deleted (that is, their destructors run and their storage is released).

2. Dynamically, where the object is created on the C++ heap. For example:

   ```
   {
   ClassX* pObjX = new ClassX;
   ClassY* pObjY = new ClassY(parameter1);
   } //objects NOT deleted here
   ```

   Here we deal with pointers to objects instead of the objects themselves. The lifetime of the object outlives the scope in which it was created. In the previous sample the pointers (pObjX and pObjY) are 'lost' as they go out of scope but the objects they pointed to still exist! The objects exist until they are explicitly deleted as shown here:

   ```
   {
   ClassX* pObjX = new ClassX;
   ClassY* pObjY = new ClassY(parameter1);
   ⋮
   pObjX->method1();
   pObjY->method2();
   ⋮
   delete pObjX;
   delete pObjY;
   }
   ```

Most of the samples in this book use automatic storage. You are ***advised*** to use automatic storage, because you do not have remember to explicitly delete objects, but you are free to use either style for CICS C++ Foundation Class programs. For more information on Foundation Classes and storage management see “Storage management” on page 40.

© Copyright IBM Corp. 1974, 2023

Using an object

Any of the class public methods can be called on an object of that class.

Any of the class public methods can be called on an object of that class. The following example creates object *obj* and then calls method **doSomething** on it:

```
ClassY obj("TEMP1234");
obj.doSomething();
```

Alternatively, you can do this using dynamic object creation:

```
ClassY* pObj = new ClassY("parameter1");
pObj->doSomething();
```

Deleting an object

When an object is destroyed its destructor function, which has the same name as the class preceded with ~(tilde), is automically called. (You cannot call the destructor explicitly).

If the object was created automatically it is automatically destroyed when it goes out of scope.

If the object was created dynamically it exists until an explicit **delete** operator is used.

Overview of the foundation classes

This topic is a formal introduction to what the Foundation Classes can do for you.

See ICC$HEL: C++ Hello World sample for a simple example to get you started. The section takes a brief look at the CICS C++ Foundation Class library by considering the categories in turn.

See Foundation classes reference for more detailed information on the Foundation Classes.

Every class that belongs to the CICS Foundation Classes is prefixed by **Icc**.

Base classes

All classes inherit, directly or indirectly, from **IccBase** .

IccBase
  IccRecordIndex
  IccResource
  IccControl
  IccTime
  IccResourceId

Figure 1. Base classes

All resource identification classes, such as **IccTermId** , and **IccTransId** , inherit from **IccResourceId** class. These are typically CICS table entries.

All CICS resources—in fact any class that needs access to CICS services—inherit from **IccResource** class.

Base classes enable common interfaces to be defined for categories of class. They are used to create the foundation classes, as provided by IBM® , and they can be used by application programmers to create their own derived classes.

IccBase
The base for every other foundation class. It enables memory management and allows objects to be interrogated to discover which type they are.

IccControl
The abstract base class that the application program has to subclass and provide with an implementation of the **run** method.

IccResource
The base class for all classes that access CICS resources or services. See “Resource classes” on page 8.

IccResourceId
The base class for all table entry (resource name) classes, such as **IccFileId** and **IccTempStoreId**.

IccTime
The base class for the classes that store time information: **IccAbsTime** , **IccTimeInterval** and **IccTimeOfDay**.

Resource identification classes

Resource identification classes are as follows.

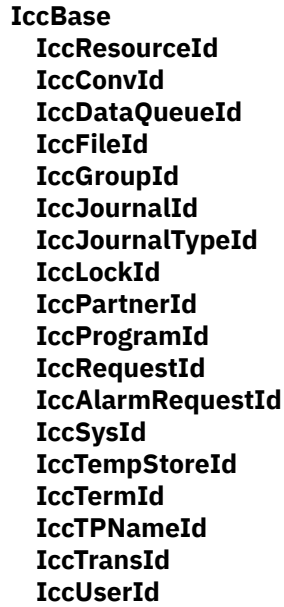


Figure 2. Resource identification classes

CICS resource identification classes define CICS resource identifiers – typically the name of the resource as specified in its RDO resource definition. For example an **IccFileId** object represents a CICS file name. All concrete resource identification classes have the following properties:

- The name of the class ends in **Id**.
- The class is a subclass of the **IccResourceId** class.
- The constructors check that any supplied resource identifier meets CICS standards. For example, an **IccFileId** object must contain a 1 to 8 byte character field; providing a 9-byte field is not tolerated.

The resource identification classes improve type checking; methods that expect an **IccFileId** object as a parameter do not accept an **IccProgramId** object instead. If character strings representing the resource names are used instead, the compiler cannot check for validity – it cannot check whether the string is a file name or a program name.

Many of the resource classes, described in “Resource classes” on page 8 , contain resource identification classes. For example, an **IccFile** object contains an **IccFileId** object. You must use the resource object, not the resource identification object, to operate on a CICS resource. For example, you must use **IccFile** , rather than **IccFileId** to read a record from a file.

Class	CICS resource
IccAlarmRequestId	alarm request
IccConvId	conversation
IccDataQueueId	transient data queue
IccFileId	file
IccGroupId	group
IccJournalId	journal
IccJournalTypeId	journal type
IccLockId	(Not applicable)
IccPartnerId	APPC partner definition files
IccProgramId	program
IccRequestId	request
IccSysId	remote system
IccTempStoreId	temporary storage queue
IccTermId	terminal
IccTPNameId	remote APPC TP name
IccTransId	transaction
IccUserId	user

Resource classes

All CICS resource classes inherit from the **IccResource** base class.

IccBase
- **IccResource**
- **IccAbendData**
- **IccClock**
- **IccConsole**
- **IccControl**
- **IccDataQueue**
- **IccFile**
- **IccFileIterator**
- **IccJournal**
- **IccProgram**
- **IccSemaphore**
- **IccSession**
- **IccStartRequestQ**
- **IccSystem**
- **IccTask**
- **IccTempStore**
- **IccTerminal**
- **IccTerminalData**
- **IccUser**

Figure 3. Resource classes

These classes model the behavior of the major CICS resources, for example:

- Terminals are modelled by **IccTerminal**.
- Programs are modelled by **IccProgram**.
- Temporary Storage queues are modelled by **IccTempStore**.
- Transient Data queues are modelled by **IccDataQueue**.

Any operation on a CICS resource may raise a CICS condition; the **condition** method of **IccResource** (see IccResource method: condition) can interrogate it.

(Any class that accesses CICS services ***must*** be derived from **IccResource**).

Class	CICS resource
IccAbendData	task abend data
IccClock	CICS time and date services
IccConsole	CICS console
IccControl	control of executing program
IccDataQueue	transient data queue
IccFile	file
IccFileIterator	file iterator (browsing files)
IccJournal	user or system journal
IccProgram	program (outside executing program)
IccSemaphore	semaphore (locking services)
IccSession	session
IccStartRequestQ	start request queue; asynchronous transaction starts
IccSystem	CICS system
IccTask	current task
IccTempStore	temporary storage queue
IccTerminal	terminal belonging to current task
IccTerminalData	attributes of **IccTerminal**
IccTime	time specification
IccUser	user (security attributes)

Support Classes

Support classes are as follows.

IccBase
- **IccBuf**
- **IccEvent**
- **IccException**
- **IccMessage**
- **IccRecordIndex**
- **IccKey**
- **IccRBA**
- **IccRRN**
- **IccResource**
- **IccTime**
- **IccAbsTime**
- **IccTimeInterval**
- **IccTimeOfDay**

Figure 4. Support classes

These classes are tools that complement the resource classes: they make life easier for the application programmer and thus add value to the object model.

Resource class	Description
IccAbsTime	Absolute time (milliseconds since January 1 1900)
IccBuf	Data buffer (makes manipulating data areas easier)
IccEvent	Event (the outcome of a CICS command)
IccException	Foundation Class exception (supports the C++ exception handling model)
IccTimeInterval	Time interval (for example, five minutes)
IccTimeOfDay	Time of day (for example, five minutes past six)

IccAbsTime, IccTimeInterval and **IccTimeOfDay** classes make it simpler for the application programmer to specify time measurements as objects within an application program. **IccTime** is a base class: **IccAbsTime** , **IccTimeInterval** , and **IccTimeOfDay** are derived from **IccTime**.

Consider method **delay** in class **IccTask** , whose signature is as follows:

```
void delay(const IccTime& time, const IccRequestId*
reqId = 0);
```

To request a delay of 1 minute and 7 seconds (that is, a time interval) the application programmer can do this:

```
IccTimeInterval time(0, 1, 7);
task()->delay(time);
```

Note: The task method is provided in class **IccControl** and returns a pointer to the application's task object.

Alternatively, to request a delay until 10 minutes past twelve (lunchtime?) the application programmer can do this:

```
IccTimeOfDay lunchtime(12, 10);
task()->delay(lunchtime);
```

The **IccBuf** class allows easy manipulation of buffers, such as file record buffers, transient data record buffers, and COMMAREAs (for more information on **IccBuf** class see “Buffer objects” on page 12).

IccMessage class is used primarily by **IccException** class to encapsulate a description of why an exception was thrown. The application programmer can also use **IccMessage** to create their own message objects.

IccException objects are thrown from many of the methods in the Foundation Classes when an error is encountered.

The **IccEvent** class allows a programmer to gain access to information relating to a particular CICS event (command).

Using CICS resources

To use a CICS resource, such as a file or program, you must first create an appropriate object and then call methods on the object.

Creating a resource object

When you create a resource object you create a representation of the actual CICS resource (such as a file or program). You do not create the CICS resource; the object is the application's view of the resource. The same is true of destroying objects.

U se an accompanying resource identification object when creating a resource object. For example:

```
IccFileId id("XYZ123");
IccFile file(id);
```

This allows the C++ compiler to protect you against doing something wrong such as:

```
IccDataQueueId id("WXYZ");
IccFile file(id); //gives error at compile time
```

The alternative of using the text name of the resource when creating the object is also permitted:

```
IccFile file("XYZ123");
```

Singleton classes

Many resource classes, such as **IccFile** , can be used to create multiple resource objects within a single program.

```
IccFileId id1("File1");
IccFileId id2("File2");
IccFile file1(id1);
IccFile file2(id2);
```

However, some resource classes are designed to allow the programmer to create only ***one*** instance of the class; these are called singleton classes. The following Foundation Classes are singleton:

- **IccAbendData** provides information about task abends.
- **IccConsole** , or a derived class, represents the system console for operator messages.
- **IccControl** , or a derived class, such as **IccUserControl** , controls the executing program.
- **IccStartRequestQ** , or a derived class, allows the application program to start CICS transactions (tasks) asynchronously.
- **IccSystem** , or a derived class, is the application view of the CICS system in which it is running.
- **IccTask** , or a derived class, represents the CICS task under which the executing program is running.
- **IccTerminal** , or a derived class, represents your task's terminal, provided that your principal facility is a 3270 terminal.

Any attempt to create more than one object of a singleton class results in an error – a C++ exception is thrown.

A class method, **instance** , is provided for each of these singleton classes, which returns a pointer to the requested object and creates one if it does not already exist. For example:

```
IccControl* pControl = IccControl::instance();
```

Calling methods on a resource object

Any of the public methods can be called on an object of that class.

For example:

```
IccTempStoreId id("TEMP1234");
IccTempStore temp(id);
temp.writeItem("Hello TEMP1234");
```

Method **writeItem** writes the contents of the string it is passed ("Hello TEMP1234") to the CICS Temporary Storage queue "TEMP1234".

Buffer objects

The Foundation Classes make extensive use of **IccBuf** objects – buffer objects that simplify the task of handling pieces of data or records.

Understanding the use of these objects is a necessary precondition for much of the rest of this book.

Each of the CICS Resource classes that involve passing data to CICS (for example by writing data records) and getting data from CICS (for example by reading data records) make use of the **IccBuf** class. Examples of such classes are **IccConsole** , **IccDataQueue** , **IccFile** , **IccFileIterator** , **IccJournal** , **IccProgram** , **IccSession** , **IccStartRequestQ** , **IccTempStore** , and **IccTerminal**.

IccBuf class

IccBuf , which is described in detail in the reference part of this book, provides generalized manipulation of data areas.

Because it can be used in a number of ways, there are several **IccBuf** constructors that affect the behavior of the object. Two important attributes of an **IccBuf** object are now described.

Data area ownership

IccBuf has an attribute indicating whether the data area has been allocated inside or outside of the object.

The possible values of this attribute are 'internal' and 'external'. It can be interrogated by using the **dataAreaOwner** method.

Internal/External ownership of buffers

When **DataAreaOwner** = external, it is the application programmer's responsibility to ensure the validity of the storage on which the **IccBuf** object is based. If the storage is invalid or inappropriate for a particular method applied to the object, unpredictable results will occur.

Data area extensibility

This attribute defines whether the length of the data area within the **IccBuf** object, once created, can be increased.

The possible values of this attribute are 'fixed' and 'extensible'. It can be interrogated by using the **dataAreaType** method.

As an object that is 'fixed' cannot have its data area size increased, the length of the data (for example, a file record) assigned to the **IccBuf** object must not exceed the data area length, otherwise a C++ exception is thrown.

Note: By definition, an 'extensible' buffer ***must*** also be 'internal'.

IccBuf constructors

There are several forms of the **IccBuf** constructor, used when creating **IccBuf** objects.

Some examples are shown here.

```
IccBuf buffer;
```

This creates an 'internal' and 'extensible' data area that has an initial length of zero. When data is assigned to the object the data area length is automatically extended to accommodate the data being assigned.

```
IccBuf buffer(50);
```

This creates an 'internal' and 'extensible' data area that has an initial length of 50 bytes. The data length is zero until data is assigned to the object. If 50 bytes of data are assigned to the object, both the data length and the data area length return a value of 50. When more than 50 bytes of data are assigned into the object, the data area length is automatically (that is, without further intervention) extended to accommodate the data.

```
IccBuf buffer(50, IccBuf::fixed);
```

This creates an 'internal' and 'fixed' data area that has a length of 50 bytes. If an attempt is made to assign more than 50 bytes of data into the object, the data is truncated and an exception is thrown to notify the application of the error situation.

```
struct MyRecordStruct
{
short id;
short code;
char data(30);
char rating;
};
MyRecordStruct myRecord;
IccBuf buffer(sizeof(MyRecordStruct), &myRecord);
```

This creates an **IccBuf** object that uses an 'external' data area called myRecord. By definition, an 'external' data area is also 'fixed'. Data can be assigned using the methods on the **IccBuf** object or using the `myRecord` structure directly.

```
IccBuf buffer("Hello World");
```

This creates an 'internal' and 'extensible' data area that has a length equal to the length of the string "Hello World". The string is copied into the object's data area. This initial data assignment can then be changed using one of the manipulation methods (such as **insert** , **cut** , or **replace**) provided.

```
IccBuf buffer("Hello World");
buffer << " out there";
IccBuf buffer2(buffer);
```

Here the copy constructor creates the second buffer with almost the same attributes as the first; the exception is the data area ownership attribute – the second object always contains an 'internal' data area that is a copy of the data area in the first. In the given example buffer2 contains "Hello World out there" and has both data area length and data length of 21.

IccBuf methods

An **IccBuf** object can be manipulated using a number of supplied methods; for example you can append data to the buffer, change the data in the buffer, cut data out of the buffer, or insert data into the middle of the buffer.

The operators **const char*** , **=** , **+=** , **==** , **!=** , and **<<** have been overloaded in class **IccBuf** . There are also methods that allow the **IccBuf** attributes to be queried. For more details see the reference section.

Working with IccResource subclasses

To illustrate working with IccResource subclasses, consider writing a queue item to CICS temporary storage using **IccTempstore** class.

```
IccTempStore store("TEMP1234");
IccBuf buffer(50);
```

The **IccTempStore** object created is the application's view of the CICS temporary storage queue named "TEMP1234". The **IccBuf** object created holds a 50-byte data area (it also happens to be 'extensible').

```
buffer = "Hello Temporary Storage Queue";
store.writeItem(buffer);
```

The character string "Hello Temporary Storage Queue" is copied into the buffer. This is possible because the **operator=** method has been overloaded in the **IccBuf** class.

The **IccTempStore** object calls its **writeItem** method, passing a reference to the **IccBuf** object as the first parameter. The contents of the **IccBuf** object are written out to the CICS temporary storage queue.

Now consider the inverse operation, reading a record from the CICS resource into the application program's **IccBuf** object:

```
buffer = store.readItem(5);
```

The **readItem** method reads the contents of the fifth item in the CICS Temporary Storage queue and returns the data as an **IccBuf** reference.

The C++ compiler resolves the given line of code into two method calls, **readItem** defined in class **IccTempStore** and **operator=** which has been overloaded in class **IccBuf** . This second method takes the contents of the returned **IccBuf** reference and copies its data into the buffer.

The given style of reading and writing records using the foundation classes is typical. The final example shows how to write code – using a similar style to the above example – but this time accessing a CICS transient data queue.

```
IccDataQueue queue("DATQ");
IccBuf buffer(50);
buffer = queue.readItem();
buffer << "Some extra data";
queue.writeItem(buffer);
```

The **readItem** method of the **IccDataQueue** object is called, returning a reference to an **IccBuf** which it then assigns (via **operator=** method, overloaded in class **IccBuf**) to the `buffer` object. The character string – "Some extra data" – is appended to the buffer (via **operator chevron «** method, overloaded in class **IccBuf**). The **writeItem** method then writes back this modified buffer to the CICS transient data queue.

You can find further examples of this syntax in the samples presented in the following sections, which describe how to use the foundation classes to access CICS services.

Refer to the reference section for further information on the **IccBuf** class. You might also find the supplied sample – `ICC$BUF` – helpful.

Using CICS Services

This section describes how to use CICS services. The services are considered in turn.

File control

The file control classes **IccFile** , **IccFileId** , **IccKey** , **IccRBA** , and **IccRRN** allow you to read, write, update and delete records in files.

In addition, **IccFileIterator** class allows you to browse through all the records in a file.

An **IccFile** object is used to represent a file. It is convenient, but not necessary, to use an **IccFileId** object to identify a file by name.

An application program reads and writes its data in the form of individual records. Each read or write request is made by a method call. To access a record, the program must identify both the file and the particular record.

VSAM (or VSAM-like) files are of the following types:

KSDS
Key-sequenced: each record is identified by a key – a field in a predefined position in the record. Each key must be unique in the file.

The logical order of records within a file is determined by the key. The physical location is held in an index which is maintained by VSAM.

When browsing, records are found in their logical order.

ESDS
Entry-sequenced: each record is identified by its relative byte address (RBA).

Records are held in an ESDS in the order in which they were first loaded into the file. New records are always added at the end and records may not be deleted or have their lengths altered.

When browsing, records are found in the order in which they were originally written.

RRDS file
Relative record: records are written in fixed-length slots. A record is identified by the relative record number (RRN) of the slot which holds it.

Reading records

A read operation uses two classes – **IccFile** to perform the operation and one of **IccKey** , **IccRBA** , and **IccRRN** to identify the particular record, depending on whether the file access type is KSDS, ESDS, or RRDS.

The **readRecord** method of **IccFile** class reads the record.

Reading KSDS records

Before reading a record you must use the **registerRecordIndex** method of **IccFile** to associate an object of class **IccKey** with the file.

You must use a key, held in the **IccKey** object, to access records. A 'complete' key is a character string of the same length as the physical file's key. Every record can be separately identified by its complete key.

A key can also be 'generic'. A generic key is shorter than a complete key and is used for searching for a set of records. The **IccKey** class has methods that allow you to set and change the key.

IccFile class has methods **isReadable** , **keyLength** , **keyPosition** , **recordIndex** , and **recordLength** , which help you when reading KSDS records.

Reading ESDS records

You must use a relative byte address (RBA) held in an **IccRBA** object to access the beginning of a record.

Before reading a record you must use the **registerRecordIndex** method of **IccFile** to associate an object of class **IccRBA** with the file.

IccFile class has methods **isReadable** , **recordFormat** , **recordIndex** , and **recordLength** that help you when reading ESDS records.

Reading RRDS records

You must use a relative record number (RRN) held in an **IccRRN** object to access a record.

Before reading a record you must use **registerRecordIndex** method of **IccFile** to associate an object of class **IccRRN** with the file.

IccFile class has methods **isReadable** , **recordFormat** , **recordIndex** , and **recordLength** which help you when reading RRDS records.

Writing records

Writing records is also known as "adding records".

This topic describes writing records that have not previously been written. Writing records that already exist is not permitted unless they have been previously been put into 'update' mode. See “Updating records” on page 17 for more information.

Before writing a record you must use **registerRecordIndex** method of **IccFile** to associate an object of class **IccKey** , **IccRBA** , or **IccRRN** with the file. The **writeRecord** method of **IccFile** class writes the record.

A write operation uses two classes – **IccFile** to perform the operation and one of **IccKey** , **IccRBA** , and **IccRRN** to identify the particular record, depending on whether the file access type is KSDS, ESDS, or RRDS.

If you have more than one record to write, you can improve the speed of writing by using mass insertion of data. You begin and end this mass insertion by calling the **beginInsert** and **endInsert** methods of **IccFile**.

Writing KSDS records

You must use a key, held in an **IccKey** object to access records.

A 'complete' key is a character string that uniquely identifies a record. Every record can be separately identified by its complete key.

The **writeRecord** method of **IccFile** class writes the record.

IccFile class has methods **isAddable** , **keyLength** , **keyPosition** , **recordIndex** , **recordLength** , and **registerRecordIndex** which help you when writing KSDS records.

Writing ESDS records

You must use a relative byte address (RBA) held in an **IccRBA** object to access the beginning of a record.

IccFile class has methods **isAddable** , **recordFormat** , **recordIndex** , **recordLength** , and **registerRecordIndex** that help you when writing ESDS records.

Writing RRDS records

Use the **writeRecord** method to add a new ESDS record.

IccFile class has methods **isAddable** , **recordFormat** , **recordIndex** , **recordLength** , and **registerRecordIndex** that help you when writing RRDS records.

Updating records

Updating a record is also known as "rewriting a record".

Before updating a record you must first read it, using **readRecord** method in 'update' mode. This locks the record so that nobody else can change it.

Use **rewriteRecord** method to update the record. Note that the **IccFile** object remembers which record is being processed and this information is not passed in again.

For an example, see code fragment: "Read record for update".

The base key in a KSDS file must not be altered when the record is modified. If the file definition allows variable-length records, the length of the record can be changed.

The length of records in an ESDS, RRDS, or fixed-length KSDS file must not be changed on update.

For a file defined to CICS as containing fixed-length records, the length of record being updated must be the same as the original length. The length of an updated record must not be greater than the maximum defined to VSAM.

Deleting records

Records can never be deleted from an ESDS file.

Deleting normal records

The **deleteRecord** method of **IccFile** class deletes one or more records, provided they are not locked by virtue of being in 'update' mode.

The records to be deleted are defined by the **IccKey** or **IccRRN** object.

Deleting locked records

The **deleteLockedRecord** method of **IccFile** class deletes a record which has been previously locked by virtue of being put in 'update' mode by the **readRecord** method.

Browsing records

Browsing, or sequential reading of files uses another class – **IccFileIterator** .

An object of this class must be associated with an **IccFile** object and an **IccKey** , **IccRBA** , or **IccRRN** object. After this association has been made the **IccFileIterator** object can be used without further reference to the other objects.

Browsing can be done either forwards, using **readNextRecord** method or backwards, using **readPreviousRecord** method. The **reset** method resets the **IccFileIterator** object to point to the record specified by the **IccKey** or **IccRBA** object.

Examples of browsing files are shown in page Code fragment "List all records in assending order of key".

Example of file control

This sample program demonstrates how to use the **IccFile** and **IccFileIterator** classes.

The source for this sample can be found in C++ sample programs , in file `ICC$FIL`. Here the code is presented without any of the terminal input and output that can be found in the source file.

```
#include "icceh.hpp"
#include "iccmain.hpp"
```

The first two lines include the header files for the Foundation Classes and the standard **main** function which sets up the operating environment for the application program.

```
const char* fileRecords[] =
{
//NAME KEY PHONE USERID
"BACH, J S 003 00-1234 BACH ",
"BEETHOVEN, L 007 00-2244 BEET ",
"CHOPIN, F 004 00-3355 CHOPIN ",
"HANDEL, G F 005 00-4466 HANDEL ",
"MOZART, W A 008 00-5577 WOLFGANG "
};
```

This defines several lines of data that are used by the sample program.

```
void IccUserControl::run()
{
```

The **run** method of **IccUserControl** class contains the user code for this example. As a terminal is to be used, the example starts by creating a terminal object and clearing the associated screen.

```
 short recordsDeleted = 0;
IccFileId id("ICCKFILE");
IccKey key(3,IccKey::generic);
IccFile file( id );
file.registerRecordIndex( &key );
key = "00";
recordsDeleted = file.deleteRecord();
```

The *key* and *file* objects are first created and then used to delete all the records whose key starts with "00" in the KSDS file "ICCKFILE". *key* is defined as a generic key having 3 bytes, only the first two of which are used in this instance.

```
 IccBuf buffer(40);
                        key.setKind( IccKey::complete );
                        for (short j = 0; j < 5; j++)
                        {
                        buffer = fileRecords[j];
                        key.assign(3, fileRecords[j]+15);
                        file.writeRecord( buffer );
                        }
```

This next fragment writes all the data provided into records in the file. The data is passed by means of an **IccBuf** object that is created for this purpose. **setKind** method is used to change *key* from 'generic' to 'complete'.

The **for** loop between these calls loops round all the data, passing the data into the buffer, using the **operator=** method of **IccBuf** , and thence into a record in the file, by means of **writeRecord**. On the way the key for each record is set, using **assign** , to be a character string that occurs in the data (3 characters, starting 15 characters in).

```
 IccFileIterator fIterator( &file,
                        &key );
                        key = "000";
                        buffer = fIterator.readNextRecord();
                        while (fIterator.condition() == IccCondition::NORMAL)
                        {
                        term->sendLine("- record read: [%s]",(const char*) buffer);
                        buffer = fIterator.readNextRecord();
                        }
```

The loop shown here lists to the terminal, using **sendLine** , all the records in ascending order of key. It uses an **IccFileIterator** object to browse the records. It starts by setting the minimum value for the key which, as it happens, does not exist in this example, and relying on CICS to find the first record in key sequence.

The loop continues until any condition other than NORMAL is returned.

```
 key = "\xFF\xFF\xFF";
fIterator.reset( &key );
buffer = fIterator.readPreviousRecord();
while (fIterator.condition() == IccCondition::NORMAL)
{
buffer = fIterator.readPreviousRecord();
}
```

The next loop is nearly identical to the last, but lists the records in reverse order of key.

```
 key = "008";
                        buffer = file.readRecord( IccFile::update );
                        buffer.replace( 4, "5678", 23);
                        file.rewriteRecord( buffer );
```

This fragment reads a record for update, locking it so that others cannot change it. It then modifies the record in the buffer and writes the updated record back to the file.

```
 buffer = file.readRecord();
```

The same record is read again and sent to the terminal, to show that it has indeed been updated.

```
 return;
}
```

The end of **run** , which returns control to CICS.

See C++ sample programs for the expected output from this sample.

Program control

This section describes how to access and use a program other than the one that is currently executing.

Program control uses **IccProgram** class, one of the resource classes.

Programs may be loaded, unloaded and linked to, using an **IccProgram** object. An **IccProgram** object can be interrogated to obtain information about the program. See IccProgram class for more details.

The example shown here shows one program calling another two programs in turn, with data passing between them via a COMMAREA. One program is assumed to be local, the second is on a remote CICS system. The programs are in two files, ICC$PRG1 and ICC$PRG2. See C++ sample programs for the location of these files and the expected output from these sample programs.

Most of the terminal IO in these samples has been omitted from the code that follows.

```
#include "icceh.hpp"
#include "iccmain.hpp"
void IccUserControl::run()
{
```

The code for both programs starts by including the header files for the Foundation Classes and the stub for **main** method. The user code is located in the **run** method of the **IccUserControl** class for each program.

```
IccSysId sysId( "ICC2" );
IccProgram icc$prg2( "ICC$PRG2" );
IccProgram remoteProg( "ICC$PRG3" );
IccBuf commArea( 100, IccBuf::fixed );
```

The first program (ICC$PRG1) creates an **IccSysId** object representing the remote region, and two **IccProgram** objects representing the local and remote programs that will be called from this program. A 100 byte, fixed length buffer object is also created to be used as a communication area between programs.

```
icc$prg2.load();
if (icc$prg2.condition() == IccCondition::NORMAL)
{
term->sendLine( "Loaded program: %s <%s> Length=%ld Address=%x",
icc$prg2.name(),
icc$prg2.conditionText(),
icc$prg2.length(),
icc$prg2.address() );
icc$prg2.unload();
}
```

The program then attempts to load and interrogate the properties of program ICC$PRG2.

```
 commArea = "DATA SET BY ICC$PRG1";
icc$prg2.link( &commArea );
```

The communication area buffer is set to contain some data to be passed to the first program that ICC$PRG1 links to (ICC$PRG2). ICC$PRG1 is suspended while ICC$PRG2 is run.

The called program, ICC$PRG2, is a simple program, the gist of which is as follows:

```
IccBuf& commArea = IccControl::commArea();
commArea = "DATA RETURNED BY ICC$PRG2";
return;
```

ICC$PRG2 gains access to the communication area that was passed to it. It then modifies the data in this communication area and passes control back to the program that called it.

The first program (ICC$PRG1) now calls another program, this time on another system, as follows:

```
remoteProg.setRouteOption( sysId );
commArea = "DATA SET BY ICC$PRG1";
remoteProg.link( &commArea );
```

The **setRouteOption** requests that calls on this object are routed to the remote system. The communication area is set again (because it will have been changed by ICC$PRG2) and it then links to the remote program (ICC$PRG3 on system ICC2).

The called program uses CICS temporary storage but the three lines we consider are:

```
IccBuf& commArea = IccControl::commArea();
commArea = "DATA RETURNED BY ICC$PRG3";
return;
```

Again, the remote program (ICC$PRG3) gains access to the communication area that was passed to it. It modifies the data in this communication area and passes control back to the program that called it.

```
 return;
};
```

Finally, the calling program itself ends and returns control to CICS.

Starting transactions asynchronously

The **IccStartRequestQ** class enables a program to start another CICS transaction instance asynchronously (and optionally pass data to the started transaction).

The same class is used by a started transaction to gain access to the data that the task that issued the start request passed to it. Finally start requests (for some time in the future) can be cancelled.

Starting transactions

You can use any of the following methods to establish what data will be sent to the started transaction.

- **registerData** or **setData**
- **setQueueName**
- **setReturnTermId**

- **setReturnTransId**

The actual start is requested using the **start** method.

Accessing start data

A started transaction can access its start data by invoking the **retrieveData** method.

This method stores all the start data attributes in the **IccStartRequestQ** object such that the individual attributes can be accessed using the following methods:

- **data**
- **queueName**
- **returnTermId**
- **returnTransId**

Cancelling unexpired start requests

Unexpired start requests (that is, start requests for some future time that has not yet been reached) can be cancelled using the **cancel** method.

Example of starting transactions

start transaction ISR1 on terminal PEO1 on system ICC1.

CICS system	**ICC1**	**ICC2**
Transaction	ISR1/ITMP	ISR2
Program	ICC$SRQ1/ICC$TMP	ICC$SRQ2
Terminal	PEO1	PEO2

This issues two start requests; the first is cancelled before it has expired. The second starts transaction ISR2 on terminal PEO2 on system ICC2. This transaction accesses its start data and finishes by starting transaction ITMP on the original terminal (PEO1 on system ICC1).

The programs and the expected output from them, can be found in C++ sample programs as files ICC$SRQ1 and ICC$SRQ2. Here the code is presented without the terminal IO requests.

Transaction ISR1 runs program ICC$SRQ1 on system ICC1. Let us consider this program first:

```
#include "icceh.hpp"
#include "iccmain.hpp"
void IccUserControl::run()
{
```

These lines include the header files for the Foundation Classes, and the **main** function needed to set up the class library for the application program. The **run** method of **IccUserControl** class contains the user code for this example.

```
 IccRequestId req1;
IccRequestId req2("REQUEST1");
IccTimeInterval ti(0,0,5);
IccTermId remoteTermId("PE02");
IccTransId ISR2("ISR2");
IccTransId ITMP("ITMP");
IccBuf buffer;
IccStartRequestQ* startQ = startRequestQ();
```

Here we are creating a number of objects:

req1
An empty **IccRequestId** object ready to identify a particular start request.

req2
An **IccRequestId** object containing the user-supplied identifier "REQUEST1".

ti
An **IccTimeInterval** object representing 0 hours, 0 minutes, and 5 seconds.

remoteTermId
An **IccTermId** object; the terminal on the remote system where we start a transaction.

ISR2
An **IccTransId** object; the transaction we start on the remote system.

ITMP
An **IccTransId** object; the transaction that the started transaction starts on this program's terminal.

buffer
An **IccBuf** object that holds start data.

Finally, the **startRequestQ** method of **IccControl** class returns a pointer to the single instance (singleton) class **IccStartRequestQ**.

```
 startQ->setRouteOption( "ICC2" );
startQ->registerData( &buffer );
startQ->setReturnTermId( terminal()->name() );
startQ->setReturnTransId( ITMP );
startQ->setQueueName( "startqnm" );
```

This code fragment prepares the start data that is passed when we issue a start request. The **setRouteOption** says we will issue the start request on the remote system, ICC2. The **registerData** method associates an **IccBuf** object that will contain the start data (the contents of the **IccBuf** object are not extracted until we issue the start request). The **setReturnTermId** and **setReturnTransId** methods allow the start requester to pass a transaction and terminal name to the started transaction. These fields are typically used to allow the started transaction to start ***another*** transaction (as specified) on another terminal, in this case ours.

The **setQueueName** is another piece of information that can be passed to the started transaction.

```
 buffer = "This is a greeting from program
'icc$srq1'!!";
req1 = startQ->start( ISR2, &remoteTermId, &ti );
startQ->cancel( req1 );
```

Here we set the data that we pass on the start requests. We start transaction ISR2 after an interval *ti* (5 seconds). The request identifier is stored in *req1* . Before the five seconds has expired (that is, immediately) we cancel the start request.

```
 req1 = startQ->start( ISR2, &remoteTermID,
&ti, &req2 );
return;
}
```

Again we start transaction ISR2 after an interval *ti* (5 seconds). This time the request is allowed to expire so transaction ISR2 is started on the remote system. Meanwhile, we end by returning control to CICS.

Let us now consider the started program, ICC$SRQ2.

```
 IccBuf buffer;
IccRequestId req("REQUESTX");
IccTimeInterval ti(0,0,5);
IccStartRequestQ* startQ = startRequestQ();
```

Here, as in ICC$SRQ1 , we create a number of objects:

buffer
An **IccBuf** object to hold the start data we were passed by our caller (ICC$SRQ1).

req
An **IccRequestId** object to identify the start we will issue on our caller's terminal.

ti
An **IccTimeInterval** object representing 0 hours, 0 minutes, and 5 seconds.

The **startRequestQ** method of **IccControl** class returns a pointer to the singleton class **IccStartRequestQ**.

```
 if ( task()->startType() != IccTask::startRequest )
{
term->sendLine(
"This program should only be started via the StartRequestQ");
task()->abend( "OOPS" );
}
```

Here we use the **startType** method of **IccTask** class to check that ICC$SRQ2 was started by the **start** method, and not in any other way (such as typing the transaction name on a terminal). If it was not started as intended, we abend with an "OOPS" abend code.

```
 startQ->retrieveData();
```

We retrieve the start data that we were passed by ICC$SRQ1 and store within the **IccStartRequestQ** object for subsequent access.

```
 buffer = startQ->data();
term->sendLine( "Start buffer contents = [%s]", buffer.dataArea() );
term->sendLine( "Start queue= [%s]", startQ->queueName() );
term->sendLine( "Start rtrn = [%s]",
startQ->returnTransId().name());
term->sendLine( "Start rtrm = [%s]", startQ->returnTermId().name() );
```

The start data buffer is copied into our **IccBuf** object. The other start data items (queue, returnTransId, and returnTermId) are displayed on the terminal.

```
 task()->delay( ti );
```

We delay for five seconds (that is, we sleep and do nothing).

```
 startQ->setRouteOption( "ICC1" );
```

The **setRouteOption** signals that we will start on our caller's system (ICC1).

```
 startQ->start(
startQ->returnTransId(),startQ->returnTermId());
return;
```

We start a transaction called ITMP (the name of which was passed by ICC$SRQ1 in the returnTransId start information) on the originating terminal (where ICC$SRQ1 completed as it started this transaction). Having issued the start request, ICC$SRQ1 ends, by returning control to CICS.

Finally, transaction ITMP runs on the first terminal. This is the end of this demonstration of starting transactions asynchronously.

Transient Data

The transient data classes, **IccDataQueue** and **IccDataQueueId** , allow you to store data in transient data queues for subsequent processing.

You can:

- Read data from a transient data queue (**readItem** method)
- Write data to a transient data queue (**writeItem** method)
- Delete a transient data queue (**empty** method)

An **IccDataQueue** object is used to represent a temporary storage queue. An **IccDataQueueId** object is used to identify a queue by name. Once the **IccDataQueueId** object is initialized it can be used to identify

the queue as an alternative to using its name, with the advantage of additional error detection by the C++ compiler.

The methods available in **IccDataQueue** class are similar to those in the **IccTempStore** class. For more information on these see “Temporary storage” on page 25.

Reading data

The **readItem** method is used to read items from the queue.

It returns a reference to the **IccBuf** object that contains the information.

Writing data

The **writeItem** method of **IccDataQueue** adds a new item of data to the queue, taking the data from the buffer specified.

Deleting queues

The **empty** method deletes all items on the queue.

Example of managing transient data

This sample program demonstrates how to use the **IccDataQueue** and **IccDataQueueId** classes.

It can be found, along with the expected output, in C++ sample programs as file `ICC$DAT`. Here the code is presented without the terminal IO requests.

```
#include "icceh.hpp"
#include "iccmain.hpp"
```

The first two lines include the header files for the foundation classes and the standard **main** function that sets up the operating environment for the application program.

```
const char* queueItems[] =
{
"Hello World - item 1",
"Hello World - item 2",
"Hello World - item 3"
};
```

This defines some buffer for the sample program.

```
void IccUserControl::run()
{
```

The **run** method of **IccUserControl** class contains the user code for this example.

```
 short itemNum =1;
IccBuf buffer( 50 );
IccDataQueueId id( "ICCQ" );
IccDataQueue queue( id );
queue.empty();
```

This fragment first creates an identification object, of type IccDataQueueId containing "ICCQ". It then creates an **IccDataQueue** object representing the transient data queue "ICCQ", which it empties of data.

```
 for (short i=0 ; i<3 ; i++)
{
buffer = queueItems[i];
queue.writeItem( buffer );
}
```

This loop writes the three data items to the transient data object. The data is passed by means of an **IccBuf** object that was created for this purpose.

```
 buffer = queue.readItem();
while ( queue.condition() == IccCondition::NORMAL )
{
buffer = queue.readItem();
}
```

Having written out three records we now read them back in to show they were successfully written.

```
 return;
}
```

The end of **run** , which returns control to CICS.

Temporary storage

The temporary storage classes, **IccTempStore** and **IccTempStoreId** , allow you to store data in temporary storage queues.

You can:

- Read an item from the temporary storage queue (**readItem** method)
- Write a new item to the end of the temporary storage queue (**writeItem** method)
- Update an item in the temporary storage queue (**rewriteItem** method)
- Read the next item in the temporary storage queue (**readNextItem** method)
- Delete all the temporary data (**empty** method)

An **IccTempStore** object is used to represent a temporary storage queue. An **IccTempStoreId** object is used to identify a queue by name. Once the **IccTempStoreId** object is initialized it can be used to identify the queue as an alternative to using its name, with the advantage of additional error detection by the C++ compiler.

The methods available in **IccTempStore** class are similar to those in the **IccDataQueue** class. For more information on these see “Transient Data” on page 23.

Reading items

The **readItem** method of **IccTempStore** reads the specified item from the temporary storage queue.

It returns a reference to the **IccBuf** object that contains the information.

Writing items

Writing items is also known as "adding" items.

This section describes writing items that have not previously been written. Writing items that already exist can be done using the **rewriteItem** method. See “Updating items” on page 25 for more information.

The **writeItem** method of **IccTempStore** adds a new item at the end of the queue, taking the data from the buffer specified. If this is done successfully, the item number of the record added is returned.

Updating items

Updating an item is also known as "rewriting" an item.

The **rewriteItem** method of **IccTempStore** class is used to update the specified item in the temporary storage queue.

Deleting items

You cannot delete individual items in a temporary storage queue.

To delete *all* the temporary data associated with an **IccTempStore** object use the **empty** method of **IccTempStore** class.

Example of Temporary Storage

This sample program demonstrates how to use the **IccTempStore** and **IccTempStoreId** classes.

This program, and the expected output from it, can be found in C++ sample programs , as file ICC$TMP. The sample is presented here without the terminal IO requests.

```
#include "icceh.hpp"
#include "iccmain.hpp"
#include <stdlib.h>
```

The first three lines include the header files for the foundation classes, the standard **main** function that sets up the operating environment for the application program, and the standard library.

```
const char* bufferItems[] =
{
"Hello World - item 1",
"Hello World - item 2",
"Hello World - item 3"
};
```

This defines some buffer for the sample program.

```
void IccUserControl::run()
{
```

The **run** method of **IccUserControl** class contains the user code for this example.

```
 short itemNum = 1;
IccTempStoreId id("ICCSTORE");
IccTempStore store( id );
IccBuf buffer( 50 );
store.empty();
```

This fragment first creates an identification object, **IccTempStoreId** containing the field "ICCSTORE". It then creates an **IccTempStore** object representing the temporary storage queue "ICCSTORE", which it empties of records.

```
 for (short j=1 ; j <= 3 ; j++)
{
buffer = bufferItems[j-1];
store.writeItem( buffer );
}
```

This loop writes the three data items to the Temporary Storage object. The data is passed by means of an **IccBuf** object that was created for this purpose.

```
 buffer = store.readItem( itemNum );
while ( store.condition() == IccCondition::NORMAL )
{
buffer.insert( 9, "Modified " );
store.rewriteItem( itemNum, buffer );
itemNum++;
buffer = store.readItem( itemNum );
}
```

This next fragment reads the items back in, modifies the item, and rewrites it to the temporary storage queue. First, the **readItem** method is used to read the buffer from the temporary storage object. The data in the buffer object is changed using the **insert** method of **IccBuf** class and then the **rewriteItem** method overwrites the buffer. The loop continues with the next buffer item being read.

```
 itemNum = 1;
buffer = store.readItem( itemNum );
while ( store.condition() == IccCondition::NORMAL )
{
term->sendLine( " - record #%d = [%s]", itemNum,
(const char*)buffer );
buffer = store.readNextItem();
}
```

This loop reads the temporary storage queue items again to show they have been updated.

```
 return;
}
```

The end of **run** , which returns control to CICS.

Terminal control

The terminal control classes, **IccTerminal** , **IccTermId** , and **IccTerminalData** , allow you to send data to, receive data from, and find out information about the terminal belonging to the CICS task.

An **IccTerminal** object is used to represent the terminal that belongs to the CICS task. It can only be created if the transaction has a 3270 terminal as its principal facility. The **IccTermId** class is used to identify the terminal. **IccTerminalData** , which is owned by **IccTerminal** , contains information about the terminal characteristics.

Sending data to a terminal

The **send** and **sendLine** methods of **IccTerminal** class are used to write data to the screen.

The **set...** methods allow you to do this. You may also want to erase the data currently displayed at the terminal, using the **erase** method, and free the keyboard so that it is ready to receive input, using the **freeKeyboard** method.

Receiving data from a terminal

The **receive** and **receive3270data** methods of **IccTerminal** class are used to receive data from the terminal.

Finding out information about a terminal

You can find out information about both the characteristics of the terminal and its current state.

The **data** object points to the **IccTerminalData** object that contains information about the characteristics of the terminal. The methods in **IccTerminalData** allow you to discover, for example, the height of the screen or whether the terminal supports Erase Write Alternative. Some of the methods in **IccTerminal** also give you information about characteristics, such as how many lines a screen holds.

Other methods give you information about the current state of the terminal. These include **line** , which returns the current line number, and **cursor** , which returns the current cursor position.

Example of terminal control

This sample program demonstrates how to use the **IccTerminal** , **IccTermId** , and **IccTerminalData** classes.

This program, and the expected output from it, can be found in C++ sample programs , as file `ICC$TRM`.

```
#include "icceh.hpp"
#include "iccmain.hpp"
```

The first two lines include the header files for the Foundation Classes and the standard **main** function that sets up the operating environment for the application program.

```
void IccUserControl::run()
{
IccTerminal& term = *terminal();
term.erase();
```

The **run** method of **IccUserControl** class contains the user code for this example. As a terminal is to be used, the example starts by creating a terminal object and clearing the associated screen.

```
 term.sendLine( "First part of the line..." );
term.send( "... a continuation of the line." );
term.sendLine( "Start this on the next line" );
term.sendLine( 40, "Send this to column 40 of current line" );
term.send( 5, 10, "Send this to row 5, column 10" );
term.send( 6, 40, "Send this to row 6, column 40" );
```

This fragment shows how the **send** and **sendLine** methods are used to send data to the terminal. All of these methods can take **IccBuf** references (const IccBuf&) instead of string literals (const char*).

```
 term.setNewLine();
```

This sends a blank line to the screen.

```
 term.setColor( IccTerminal::red );
term.sendLine( "A Red line of text.");
term.setColor( IccTerminal::blue );
term.setHighlight( IccTerminal::reverse );
term.sendLine( "A Blue, Reverse video line of text.");
```

The **setColor** method is used to set the color of the text on the screen and the **setHighlight** method to set the highlighting.

```
 term << "A cout sytle interface... " <<
endl;
term << "you can " << "chain input together; "
<< "use different types, eg numbers: " << (short)123 <<
" "
<< (long)4567890 << " " << (double)123456.7891234
<< endl;
term << "... and everything is buffered till you issue a flush."
<< flush;
```

This fragment shows how to use the iostream–like interface **endl** to start data on the next line. To improve performance, you can buffer data in the terminal until **flush** is issued, which sends the data to the screen.

```
 term.send( 24,1, "Program 'icc$trm' complete: Hit PF12
to End" );
term.waitForAID( IccTerminal::PF12 );
term.erase();
```

The **waitForAID** method causes the terminal to wait until the specified key is hit, before calling the **erase** method to clear the display.

```
 return;
}
```

The end of **run** , which returns control to CICS.

Time and date services

The **IccClock** class controls access to the CICS time and date services.

IccAbsTime holds information about absolute time (the time in milliseconds that have elapsed since the beginning of 1900), and this can be converted to other forms of date and time. The methods available on **IccClock** objects and on **IccAbsTime** objects are very similar.

Example of time and date services

This sample program demonstrates how to use **IccClock** class.

The source for this program, and the expected output from it, can be found in C++ sample programs , as file ICC$CLK . The sample is presented here without the terminal IO requests.

```
#include "icceh.hpp"
#include "iccmain.hpp"
void IccUserControl::run()
{
```

The first two lines include the header files for the Foundation Classes and the standard **main** function that sets up the operating environment for the application program.

The **run** method of **IccUserControl** class contains the user code for this example.

```
 IccClock clock;
```

This creates a clock object.

```
 term->sendLine( "date() = [%s]",
clock.date() );
term->sendLine( "date(DDMMYY) = [%s]",
clock.date(IccClock::DDMMYY) );
term->sendLine( "date(DDMMYY,':') = [%s]",
clock.date(IccClock::DDMMYY,':'));
term->sendLine( "date(MMDDYY) = [%s]",
clock.date(IccClock::MMDDYY));
term->sendLine( "date(YYDDD) = [%s]",
clock.date(IccClock::YYDDD));
```

Here the **date** method is used to return the date in the format specified by the *format* enumeration. In order the formats are system, DDMMYY, DD:MM:YY, MMDDYY and YYDDD. The character used to separate the fields is specified by the *dateSeparator* character (that defaults to nothing if not specified).

```
 term->sendLine( "daysSince1900() = %ld",
clock.daysSince1900());
term->sendLine( "dayOfWeek() = %d",
clock.dayOfWeek());
if ( clock.dayOfWeek() == IccClock::Friday )
term->sendLine( 40, "Today IS Friday" );
else
term->sendLine( 40, "Today is NOT Friday" );
```

This fragment demonstrates the use of the **daysSince1900** and **dayOfWeek** methods. **dayOfWeek** returns an enumeration that indicates the day of the week. If it is Friday, a message is sent to the screen, 'Today IS Friday'; otherwise the message 'Today is NOT Friday' is sent.

```
 term->sendLine( "dayOfMonth() = %d",
clock.dayOfMonth());
term->sendLine( "monthOfYear() = %d",
clock.monthOfYear());
```

This demonstrates the **dayOfMonth** and **monthOfYear** methods of **IccClock** class.

```
 term->sendLine( "time() = [%s]",
clock.time() );
term->sendLine( "time('-') = [%s]",
clock.time('-') );
term->sendLine( "year() = [%ld]",
clock.year());
```

The current time is sent to the terminal, first without a separator (that is HHMMSS format), then with '-' separating the digits (that is, HH-MM-SS format). The year is sent, for example 1996.

```
 return;
};
```

The end of **run** , which returns control to CICS.

Compiling, executing, and debugging

This section describes how to compile, execute, and debug a CICS Foundation Class program.

Compiling Programs

To compile and link a CICS Foundation Class program you need access to the program source, a compiler, header files and a dynamic link library.

You need access to the following items:

- The source of the program you are compiling

  Your C++ program source code needs `#include` statements for the Foundation Class headers and the Foundation Class `main()` program stub:

```
#include "icceh.hpp"
#include "iccmain.hpp"
```

- The IBM C++ compiler
- The Foundation Classes header files (see “Header files” on page 1)
- The Foundation Classes dynamic link library (DLL). The `ICCFCDLL` module is in CICSTS54.CICS .SDFHLOAD.

Note that, when using the Foundation Classes, you do not need to translate the "EXEC CICS " API so the translator program should not be used.

The following sample job statements show how to compile, prelink and link a program called `ICC$HEL` :

```
//ICC$HEL JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//PROCLIB JCLLIB ORDER=(
CICSTS54.CICS
.SDFHPROC)
//ICC$HEL EXEC ICCFCCL,INFILE=
indatasetname
(ICC$HEL),OUTFILE=
outdatasetname
(ICC$HEL)
//
```

Header files

The header files are the C++ class definitions needed to compile CICS C++ Foundation Class programs.

C++ Header File	Classes Defined in this Header
ICCABDEH	IccAbendData
ICCBASEH	IccBase
ICCBUFEH	IccBuf
ICCCLKEH	IccClock
ICCCNDEH	IccCondition (struct)
ICCCONEH	IccConsole
ICCCTLEH	IccControl
ICCDATEH	IccDataQueue
ICCEH	see “1” on page 31
ICCEVTEH	IccEvent

C++ Header File	Classes Defined in this Header
ICCEXCEH	IccException
ICCFILEH	IccFile
ICCFLIEH	IccFileIterator
ICCGLBEH	Icc (struct) (global functions)
ICCJRNEH	IccJournal
ICCMSGEH	IccMessage
ICCPRGEH	IccProgram
ICCRECEH	IccRecordIndex, IccKey, IccRBA and IccRRN
ICCRESEH	IccResource
ICCRIDEH	IccResourceId + subclasses (such as IccConvId)
ICCSEMEH	IccSemaphore
ICCSESEH	IccSession
ICCSRQEH	IccStartRequestQ
ICCSYSEH	IccSystem
ICCTIMEH	IccTime, IccAbsTime, IccTimeInterval, IccTimeOfDay
ICCTMDEH	IccTerminalData
ICCTMPEH	IccTempStore
ICCTRMEH	IccTerminal
ICCTSKEH	IccTask
ICCUSREH	IccUser
ICCVALEH	IccValue (struct)

Note:

1. A single header that #includes all the listed header files is supplied as `ICCEH`
2. The file `ICCMAIN` is also supplied with the C++ header files. This contains the **main** function stub that should be used when you build a Foundation Class program.
3. Header files are located in CICSTS54.CICS .SDFHC370.

Executing Programs

To run a compiled and linked (that is, executable) Foundation Classes program you need to do the following.

1. Make the executable program available to CICS . This involves making sure the program is in a suitable directory or load library. Depending on your server, you may also need to create a CICS program definition (using CICS resource definition facilities) before you can execute the program.
2. Logon to a CICS terminal.
3. Run the program.

Program debugging

Having successfully compiled, linked, and attempted to run your Foundation Classes program, you might need to debug it.

There are three options available to help debug a CICS Foundation Classes program:

- Use a symbolic debugger
- Run the Foundation Class Program with tracing active
- Run the Foundation Class Program with the CICS Execution Diagnostic Facility

Symbolic debugger

You can use a symbolic debugger to step through the source of your CICS Foundation Classes program. Debug Tool is shipped as a feature with IBM C/C++. To debug a CICS Foundation Classes program with a symbolic debugger, compile the program with a flag that adds debugging information to your executable program. For CICS Transaction Server for z/OS, this flag is TEST(ALL).

For more information, see Debug Tool for z/OS.

Tracing

You can configure the CICS Foundation Classes to write a trace file for debugging purposes.

Exception tracing is always active. The CETR transaction controls the auxiliary and internal traces for all CICS programs including those developed using the C++ classes.

Execution diagnostic facility

You can use the Execution Diagnostic Facility (EDF) to step through your CICS program, stopping at each **EXEC CICS** call. The display screen shows the procedural **EXEC CICS** call interface rather than the CICS Foundation Class type interface.

To enable EDF, use the preprocessor macro `ICC_EDF` in your source code before including the file `ICCMAIN`.

```
 #define ICC_EDF //switch EDF on
#include "iccmain.hpp"
```

Alternatively use the appropriate flag on your compiler CPARM to declare ICC_EDF.

Conditions, errors, and exceptions

This section describes how the Foundation Classes have been designed to respond to various error situations they might encounter.

Foundation Class Abend codes

For serious errors (such as insufficient storage to create an object) the Foundation Classes immediately terminate the CICS task.

All CICS Foundation Class abend codes are of the form ACLx. If your application is terminated with an abend code starting 'ACL' then please refer to CICS messages.

C++ Exceptions and the Foundation Classes

C++ exceptions are managed using the reserved words **try** , **throw** , and **catch**.

Refer to your compiler's documentation or one of the C++ books in the bibliography for more information.

Here is sample `ICC$EXC1` (see C++ sample programs):

```
#include "icceh.hpp"
#include "iccmain.hpp"
class Test {
public:
void tryNumber( short num ) {
IccTerminal* term = IccTerminal::instance();
*term << "Number passed = " << num << endl <<
flush;
if ( num > 10 ) {
*term << ">>Out of Range - throwing exception" << endl
<< flush;
throw "!!Number is out of range!!";
}
}
};
```

The first two lines include the header files for the Foundation Classes and the standard **main** function that sets up the operating environment for the application program.

We then declare class **Test** , which has one public method, **tryNumber** . This method is implemented inline so that if an integer greater than ten is passed an exception is thrown. We also write out some information to the CICS terminal.

```
void IccUserControl::run()
{
IccTerminal* term = IccTerminal::instance();
term->erase();
*term << "This is program 'icc$exc1' ..." << endl;
try {
Test test;
test.tryNumber( 1 );
test.tryNumber( 7 );
test.tryNumber( 11 );
test.tryNumber( 6 );
}
catch( const char* exception ) {
term->setLine( 22 );
*term << "Exception caught: " << exception << endl
<< flush;
}
term->send( 24,1,"Program 'icc$exc1' complete: Hit PF12 to End" );
term->waitForAID( IccTerminal::PF12 );
term->erase();
return;
}
```

The **run** method of **IccUserControl** class contains the user code for this example.

After erasing the terminal display and writing some text, we begin our **try** block. A **try** block can scope any number of lines of C++ code.

Here we create a **Test** object and invoke our only method, **tryNumber** , with various parameters. The first two invocations (1, 7) succeed, but the third (11) causes **tryNumber** to throw an exception. The fourth **tryNumber** invocation (6) is not executed because an exception causes the program execution flow to leave the current **try** block.

We then leave the **try** block and look for a suitable **catch** block. A suitable **catch** block is one with arguments that are compatible with the type of exception being thrown (here a **char***). The **catch** block writes a message to the CICS terminal and then execution resumes at the line after the **catch** block.

The output from this CICS program is as follows:

```
This is program 'icc$exc1' ...
                    Number passed = 1
                    Number passed = 7
                    Number passed = 11
                    >>Out of Range - throwing exception
                    Exception caught: !!Number is out of range!!
                    Program 'icc$exc1' complete: Hit PF12 to End
```

The CICS C++ Foundation Classes do not throw **char*** exceptions as in the previous sample but they do throw **IccException** objects instead.

There are several types of **IccException** . The **type** method returns an enumeration that indicates the type. Here is a description of each type in turn.

objectCreationError
An attempt to create an object was invalid. This happens, for example, if an attempt is made to create a second instance of a singleton class, such as **IccTask**.

invalidArgument
A method was called with an invalid argument. This happens, for example, if an **IccBuf** object with too much data is passed to the **writeItem** method of the **IccTempStore** class by the application program.

It also happens when attempting to create a subclass of **IccResourceId** , such as **IccTermId** , with a string that is too long.

The following sample can be found in C++ sample programs , as file `ICC$EXC2` . The sample is presented here without many of the terminal IO requests.

```
#include "icceh.hpp"
#include "iccmain.hpp"
void IccUserControl::run()
{
try
{
IccTermId id1( "1234" );
IccTermId id2( "12345");
}
catch( IccException& exception )
{
terminal()->send( 21, 1, exception.summary() );
}
return;
}
```

In the previous example the first **IccTermId** object is successfully created, but the second caused an **IccException** to be thrown, because the string "12345" is 5 bytes where only 4 are allowed. See C++ sample programs for the expected output from this sample program.

invalidMethodCall
A method cannot be called. A typical reason is that the object cannot honor the call in its current state. For example, a **readRecord** call on an **IccFile** object is only honored if an **IccRecordIndex** object, to specify ***which*** record is to be read, has already been associated with the file.

CICSCondition
A CICS condition, listed in the **IccCondition** structure, has occurred in the object and the object was configured to throw an exception.

familyConformanceError
Family subset enforcement is on for this program and an operation that is not valid on all supported platforms has been attempted.

internalError
The CICS foundation classes have detected an internal error. Please call service.

CICS conditions

The CICS foundation classes provide a powerful framework for handling conditions that happen when executing an application.

Accessing a CICS resource can raise a number of CICS conditions as documented in Foundation classes reference.

A condition might represent an error or information being returned to the calling application; the deciding factor is often the context in which the condition is raised.

The application program can handle the CICS conditions in a number of ways. Each CICS resource object, such as a program, file, or data queue, can handle CICS conditions differently, if required.

A resource object can be configured to take one of the following actions for each condition it can encounter:

noAction
Manual condition handling

callHandleEvent
Automatic condition handling

throwException
Exception handling

abendTask
Severe error handling.

Manual condition handling (noAction)

This is the default action for all CICS conditions (for any resource object).

This means that the condition must be handled manually, using the **condition** method. For example:

```
IccTempStore temp("TEMP1234");
IccBuf buf(40);
temp.setActionOnCondition(IccResource::noAction,
IccCondition::QIDERR);
buf = temp.readNextItem();
switch (temp.condition())
{
case IccCondition::QIDERR:
//do whatever here
:
default:
//do something else here
}
```

Automatic condition handling (callHandleEvent)

Activate this for any CICS condition, such as QIDERR, as follows.

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::callHandleEvent,
IccCondition::QIDERR);
```

When a call to any method on object 'temp' causes CICS to raise the QIDERR condition, **handleEvent** method is automatically called. As the **handleEvent** method is only a virtual method, this call is only useful if the object belongs to a subclass of **IccTempStore** and the **handleEvent** method has been overridden.

Make a subclass of **IccTempStore** , declare a constructor, and override the **handleEvent** method.

```
class MyTempStore : public IccTempStore
{
public:
MyTempStore(const char* storeName) : IccTempStore(storeName) {}
HandleEventReturnOpt handleEvent(IccEvent& event);
};
```

Now implement the **handleEvent** method.

```
IccResource::HandleEventReturnOpt
MyTempStore::handleEvent(IccEvent& event)
{
switch (event.condition())
{
case …
:
case IccCondition::QIDERR:
//Handle QIDERR condition here.
:
//
default:
return rAbendTask;
}
}
```

This code is called for any **MyTempStore** object which is configured to 'callHandleEvent' for a particular CICS condition.

Exception handling (throwException)

Activate this for any CICS condition, such as QIDERR, as follows.

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::throwException,
IccCondition::QIDERR);
```

Exception handling is by means of the C++ exception handling model using **try** , **throw** , and **catch** . For example:

```
try
{
buf = temp.readNextItem();
...
}
catch (IccException& exception)
{
//Exception handling code
...
}
```

An exception is thrown if any of the methods inside the try block raise the QIDERR condition for object 'temp'. When an exception is thrown, C++ unwinds the stack and resumes execution at an appropriate **catch** block – it is not possible to resume within the **try** block. For a fuller example, see sample `ICC$EXC3`.

Note: Exceptions can be thrown from the Foundation Classes for many reasons other than this example – see “C++ Exceptions and the Foundation Classes” on page 32 for more details.

Severe error handling (abendTask)

This option allows CICS to terminate the task when certain conditions are raised.

Activate this for any CICS condition, such as QIDERR, as follows:

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::abendTask,
IccCondition::QIDERR);
```

If CICS raises the QIDERR condition for object 'temp' the CICS task terminates with an ACL3 abend.

Platform differences

The CICS Foundation Classes, as described here, are designed to be independent of the particular CICS platform on which they are running. There are however some differences between platforms; these, and ways of coping with them, are described here.

Note: References in this section to other CICS platforms are included for completeness. There have been Technology Releases of the CICS Foundation Classes on those platforms.

Applications can be run in one of two modes:

fsAllowPlatformVariance
: Applications written using the CICS Foundation Classes are able to access all the functions available on the target CICS server.

fsEnforce
: Applications are restricted to the CICS functions that are available across all CICS Servers (z/OS and UNIX).

The default is to allow platform variance and the alternative is to force the application to only use features which are common to all CICS platforms.

The class headers are the same for all platforms and they "support" (that is, define) all the CICS functions that are available through the Foundation Classes on any of the CICS platforms. The restrictions on each platform are documented in Foundation classes reference. Platform variations exist at:

- object level
- method level
- parameter level

Object level

Some objects are not supported on certain platforms.

For example, **IccConsole** objects cannot be created on CICS(r) for AIX® as CICS(r) for AIX(r) does not support console services.

Any attempt to create an **IccConsole** object on CICS(r) for AIX(r) causes an **IccException** object of type 'platformError' to be thrown, but would be acceptable on the other platforms

```
IccConsole* cons = console(); //No good on CICS for AIX
```

If you initialize your application with 'fsEnforce' selected (see initializeEnvironment) the previous examples both cause an **IccException** object, of type 'familyConformanceError' to be thrown on all platforms.

Unlike objects of the **IccConsole** and **IccJournal** classes, most objects can be created on any CICS server platform. However the use of the methods can be restricted. Foundation Classes: reference fully documents all platform restrictions.

Method level

Methods that run successfully on one platform can cause a problem on another platform.

Consider, for example method **programId** in the **IccControl** class:

```
void IccUserControl::run()
{
if (strcmp(programId.name(), "PROG1234") == 0)
//do something
}
```

Here method **programId** executes correctly on CICS TS for z/OS but throws an **IccException** object of type 'platformError' on CICS(r) for AIX(r).

Alternatively, if you initialize your application with family subset enforcement on (see **initializeEnvironment** function of **Icc** structure), method **programId** throws an **IccException** object of type 'familyConformanceError' on *any* CICS server platform.

Parameter level

At this level a method is supported on all platforms, but a particular positional parameter has some platform restrictions.

Consider method **abend** in **IccTask** class.

```
task()->abend();
 1

task()->abend("WXYZ");
 2

task()->abend("WXYZ", IccTask::respectAbendHandler);
 3

task()->abend("WXYZ", IccTask::ignoreAbendHandler);
 4

task()->abend("WXYZ", IccTask::ignoreAbendHandler,
 5

IccTask::suppressDump);
```

Abends 1 to 4 run successfully on all CICS server platforms.

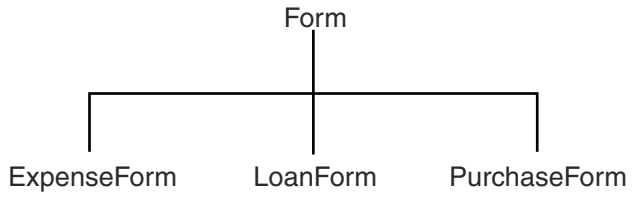
If family subset enforcement is off, abend 5 throws an **IccException** object of type 'platformError' on a CICS(r) for AIX(r) platform, but not on a CICS Transaction Server for z/OS platform.

If family subset enforcement is on, abend 5 throws an **IccException** object of type 'familyConformanceError', irrespective of the target CICS platform.

Polymorphic Behavior

Polymorphism (*poly* = many, *morphe* = form) is the ability to treat many different forms of an object as if they were the same.

Polymorphism is achieved in C++ by using inheritance and virtual functions. Consider the scenario where we have three forms (ExpenseForm, LoanForm, PurchaseForm) that are specializations of a general Form:



Each form needs printing at some time. In procedural programming, we would either code a print function to handle the three different forms or we would write three different functions (printExpenseForm, printLoanForm, printPurchaseForm).

In C++ this can be achieved far more elegantly as follows:

```
 class Form {
public:
virtual void print();
};
class ExpenseForm : public Form {
public:
virtual void print();
};
class LoanForm : public Form {
public:
virtual void print();
};
class PurchaseForm : public Form {
public:
virtual void print();
};
```

Each of these overridden functions is implemented so that each form prints correctly. Now an application using form objects can do this:

```
Form* pForm[10]
//create Expense/Loan/Purchase Forms…
for (short i=0 ; i < 9 ; i++)
pForm->print();
```

Here we create ten objects that might be any combination of Expense, Loan, and Purchase Forms. However, because we are dealing with pointers to the base class, **Form** , we do not need to know which sort of form object we have; the correct **print** method is called automatically.

Limited polymorphic behavior is available in the Foundation Classes. Three virtual functions are defined in the base class **IccResource** :

```
virtual void clear();
virtual const IccBuf& get();
virtual void put(const IccBuf&

buffer

);
```

These methods have been implemented in the subclasses of **IccResource** wherever possible:

Class	clear	get	put
IccConsole	×	×	✓
IccDataQueue	✓	✓	✓
IccJournal	×	×	✓
IccSession	×	✓	✓
IccTempStore	✓	✓	✓
IccTerminal	✓	✓	✓

These virtual methods are ***not*** supported by any subclasses of **IccResource** except those in the table.

Note: The default implementations of **clear** , **get** , and **put** in the base class **IccResource** throw an exception to prevent the user from calling an unsupported method.

Example of polymorphic behavior

The following sample can be found in the samples directory as file ICC$RES2.

It is presented here without the terminal IO requests. See C++ sample programs.

```
#include "icceh.hpp"
#include "iccmain.hpp"
char* dataItems[] =
{
"Hello World - item 1",
"Hello World - item 2",
"Hello World - item 3"
};
void IccUserControl::run()
{
```

Here we include Foundation Class headers and the **main** function. **dataItems** contains some sample data items. We write our application code in the **run** method of **IccUserControl** class.

```
 IccBuf buffer( 50 );
IccResource* pObj[2];
```

We create an **IccBuf** object (50 bytes initially) to hold our data items. An array of two pointers to **IccResource** objects is declared.

```
 pObj[0] = new IccDataQueue("ICCQ");
pObj[1] = new IccTempStore("ICCTEMPS");
```

We create two objects whose classes are derived from **IccResource** – **IccDataQueue** and **IccTempStore**.

```
 for ( short index=0; index <= 1 ; index++ )
{
pObj[index]->clear();
}
```

For both objects we invoke the **clear** method. This is handled differently by each object in a way that is transparent to the application program; this is polymorphic behavior.

```
 for ( index=0; index <= 1 ; index++ )
{
for (short j=1 ; j <= 3 ; j++)
{
buffer = dataItems[j-1];
pObj[index]->put( buffer );
}
}
```

Now we **put** three data items in each of our resource objects. Again the **put** method responds to the request in a way that is appropriate to the object type.

```
 for ( index=0; index <= 1 ; index++ )
{
buffer = pObj[index]->get();
while (pObj[index]->condition() == IccCondition::NORMAL)
{
buffer = pObj[index]->get();
}
delete pObj[index];
}
return;
}
```

The data items are read back in from each of our resource objects using the **get** method. We delete the resource objects and return control to CICS.

Storage management

C++ objects are usually stored on the stack or heap.

Objects on the stack are automatically destroyed when they go out of scope, but objects on the heap are not.

Many of the objects that the CICS Foundation Classes create internally are created on the heap rather than the stack. This can cause a problem in some CICS server environments.

On CICS Transaction Server for z/OS ,, CICS and Language Environment® manage all task storage so that it is released at task termination (normal or abnormal).

In a CICS for AIX environment, storage allocated on the heap is not automatically released at task termination. This can lead to "memory leaks" if the application programmer forgets to explicitly delete an object on the heap, or, more seriously, if the task abends.

This problem has been overcome in the CICS Foundation Classes by providing operators **new** and **delete** in the base Foundation Class, **IccBase** . These can be configured to map dynamic storage allocation requests to CICS task storage, so that ***all*** storage is automatically released at task termination. The disadvantage of this approach is a performance hit as the Foundation Classes typically issue a large number of small storage allocation requests rather than a single, larger allocation request.

This facility is affected by the **Icc::initializeEnvironment** call that must be issued before using the Foundation Classes. (This function is called from the default **main** function; see main function .)

The first parameter passed to the **initializeEnvironment** function is an enumeration that takes one of these three values:

cmmDefault
The default action is platform dependent:

z/OS
same as 'cmmNonCICS' - see the 'cmmNonCICS' section.

UNIX
same as 'cmmCICS' - see the 'cmmCICS' section.

cmmNonCICS
The **new** and **delete** operators in class **IccBase** ***do not*** map dynamic storage allocation requests to CICS task storage; instead the C++ default **new** and **delete** operators are invoked.

cmmCICS
The **new** and **delete** operators in class **IccBase** map dynamic storage allocation requests to CICS task storage (which is automatically released at normal or abnormal task termination).

The default **main** function supplied with the Foundation Classes calls **initializeEnvironment** with an enum of 'cmmDefault'. You can change this in your program without changing the supplied "header file" ICCMAIN as follows:

```
#define ICC_CLASS_MEMORY_MGMT Icc::cmmNonCICS
#include "iccmain.hpp"
```

Alternatively, set the option **DEV(ICC_CLASS_MEMORY_MGMT)** when compiling.

Parameter passing conventions

The convention used for passing objects on Foundation Classes method calls is if the object is mandatory, pass by reference; if it is optional pass by pointer.

For example, consider method **start** of class **IccStartRequestQ** , which has the following signature:

```
const IccRequestId& start( const IccTransId&
transId,
const IccTime* time=0,
const IccRequestId* reqId=0 );
```

Using the preceding convention, we see that an **IccTransId** object is mandatory, while an **IccTime** and an **IccRequestId** object are both optional. This enables an application to use this method in any of the following ways:

```
 IccTransId trn("ABCD");
IccTimeInterval int(0,0,5);
IccRequestId req("MYREQ");
IccStartRequestQ* startQ = startRequestQ();
startQ->start( trn );
startQ->start( trn, &int );
startQ->start( trn, &int, &req );
startQ->start( trn, 0, &req );
```

Scope of data in IccBuf reference returned from 'read' methods

Many of the subclasses of **IccResource** have 'read' methods that return **const IccBuf** references; for example, **IccFile::readRecord** , **IccTempStore::readItem** and **IccTerminal::receive**.

Care should be taken if you choose to maintain a reference to the **IccBuf** object, rather than copy the data from the **IccBuf** reference into your own **IccBuf** object. For example, consider the following

```
 IccBuf buf(50);
IccTempStore store("TEMPSTOR");
buf = store.readNextItem();
```

Here, the data in the **IccBuf** reference returned from **IccTempStore::readNextItem** is ***immediately*** copied into the application's own **IccBuf** object, so it does not matter if the data is later invalidated. However, the application might look like this

```
 IccTempStore store("TEMPSTOR");
const IccBuf& buf = store.readNextItem();
```

Here, the **IccBuf** reference returned from **IccTempStore::readNextItem** is ***not*** copied into the application's own storage and care must therefore be taken.

Note: You are recommended not to use this style of programming to avoid using a reference to an **IccBuf** object that does not contain valid data.

The returned **IccBuf** reference typically contains valid data until one of the following conditions is met:

- Another 'read' method is invoked on the **IccResource** object (for example, another **readNextItem** or **readItem** method in the example).
- The resource updates are committed (see method **IccTask::commitUOW**).
- The task ends (normally or abnormally).

Chapter 3. Foundation Classes: reference

This section contains the reference information on the foundation classes and structures that are provided as part of CICS. The classes and structures are arranged in alphabetic order. All the functionality you require to create object-oriented CICS programs is included within these classes and structures.

All of the classes and structures begin with the unique prefix **Icc**. Do not create your own classes with this prefix.

Icc structure contains some functions and enumerations that are widely applicable. **IccValue** structure consists of a large enumeration of all the CVDA values used in traditional CICS programs.

The description of each class starts with a simple diagram that shows how it is derived from **IccBase** class, the basis of all the other classes. This is followed by a short description and an indication of the name of the header file that includes it and, where appropriate, a sample source file that uses it.

Within each class or structure description are, where appropriate, the following sections:

1. Inheritance diagram
2. Brief description of class
3. Header file where class is defined. For the location of the C++ header files on your system see Header files.
4. Sample program demonstrating class. For the location of the supplied C++ sample programs on your system see C++ sample programs.
5. Icc... constructors
6. Public methods (in alphabetic order)
7. Protected methods (in alphabetic order)
8. Inherited public methods (in tabular form)
9. Inherited protected methods (in tabular form)
10. Enumerations

Methods, including constructors, start with a formal function prototype that shows what a call returns and what the parameters are. There follows a description, in order, of the parameters. To avoid duplication, inherited methods just have an indication of the class from which they are derived (and where they are described).

The convention for names is:

1. Variable names are shown as *variable*.
2. Names of classes, structures, enumerations and methods are shown as **method**
3. Members of enumerations are shown as 'enumMember'.
4. The names of all the supplied classes and structures begin with **Icc**.
5. Compound names have no separators, but have capital letters to demark the beginning of second and subsequent words, as in **IccJournalTypeId**.
6. Class and structure names and enumeration types begin with capital letters. Other names begin with lowercase letters.

For further information on how to use these classes, see Using the CICS foundation classes.

© Copyright IBM Corp. 1974, 2023

Mapping EXEC CICS calls to Foundation Class methods

The following table shows the correspondence between CICS calls made using the EXEC CICS API and the equivalent calls from the Foundation Classes.

EXEC CICS	Class	Method
ABEND	IccTask	abend
ADDRESS COMMAREA	IccControl	commArea
ADDRESS CWA	IccSystem	workArea
ADDRESS EIB	No direct access to EIB: please use appropriate method on appropriate class.	
ADDRESS TCTUA	IccTerminal	workArea
ADDRESS TWA	IccTask	workArea
ALLOCATE	IccSession	allocate
ASKTIME	IccClock	update
ASSIGN ABCODE	IccAbendData	abendCode
ASSIGN ABDUMP	IccAbendData	isDumpAvaliable
ASSIGN ABPROGRAM	IccAbendData	programName
ASSIGN ALTSCRNHT	IccTerminalData	alternateHeight
ASSIGN ALTSCRNWD	IccTerminalData	alternateWidth
ASSIGN APLKYBD	IccTerminalData	isAPLKeyboard
ASSIGN APLTEXT	IccTerminalData	isAPLText
ASSIGN ASRAINTRPT	IccAbendData	ASRAInterrupt
ASSIGN ASRAKEY	IccAbendData	ASRAKeyType
ASSIGN ASRAPSW	IccAbendData	ASRAPSW
ASSIGN ASRAREGS	IccAbendData	ASRARegisters
ASSIGN ASRASPC	IccAbendData	ASRASpaceType
ASSIGN ASRASTG	IccAbendData	ASRAStorageType
ASSIGN APPLID	IccSystem	applName
ASSIGN BTRANS	IccTerminalData	isBTrans
ASSIGN CMDSEC	IccTask	isCommandSecurityOn
ASSIGN COLOR	IccTerminalData	isColor
ASSIGN CWALENG	IccSystem	workArea
ASSIGN DEFSCRNHT	IccTerminalData	defaultHeight
ASSIGN DEFSCRNWD	IccTerminalData	defaultWidth
ASSIGN EWASUPP	IccTerminalData	isEWA
ASSIGN EXTDS	IccTerminalData	isExtended3270
ASSIGN FACILITY	IccTerminal	name
ASSIGN FCI	IccTask	facilityType

EXEC CICS	Class	Method
ASSIGN GCHARS	IccTerminalData	graphicCharSetId
ASSIGN GCODES	IccTerminalData	graphicCharCodeSet
ASSIGN GMMI	IccTerminalData	isGoodMorning
ASSIGN HILIGHT	IccTerminalData	isHighlight
ASSIGN INITPARM	IccControl	initData
ASSIGN INITPARMLEN	IccControl	initData
ASSIGN INVOKINGPROG	IccControl	callingProgramId
ASSIGN KATAKANA	IccTerminalData	isKatakana
ASSIGN NETNAME	IccTerminal	netName
ASSIGN OUTLINE	IccTerminalData	isFieldOutline
ASSIGN ORGABCODE	IccAbendData	originalAbendCode
ASSIGN PRINSYSID	IccTask	principalSysId
ASSIGN PROGRAM	IccControl	programId
ASSIGN PS	IccTerminalData	isPS
ASSIGN QNAME	IccTask	triggerDataQueueId
ASSIGN RESSEC	IccTask	isResourceSecurityOn
ASSIGN RESTART	IccTask	isRestarted
ASSIGN SCRNHT	IccTerminal	height
ASSIGN SCRNWD	IccTerminal	width
ASSIGN SOSI	IccTerminalData	isSOSI
ASSIGN STARTCODE	IccTask	startType, isCommitSupported, isStartDataAvailable
ASSIGN SYSID	IccSystem	sysId
ASSIGN TASKPRIORITY	IccTask	priority
ASSIGN TCTUALENG	IccTerminal	workArea
ASSIGN TEXTKYBD	IccTerminalData	isTextKeyboard
ASSIGN TEXTPRINT	IccTerminalData	isTextPrint
ASSIGN TWALENG	IccTask	workArea
ASSIGN USERID	IccTask	userId
ASSIGN VALIDATION	IccTerminalData	isValidation
CANCEL	IccClock	cancelAlarm
CANCEL	IccStartRequestQ	cancel
CHANGE PASSWORD	IccUser	changePassword
CHANGE TASK	IccTask	setPriority
CONNECT PROCESS	IccSession	connectProcess
CONVERSE	IccSession	converse

EXEC CICS	Class	Method
DELAY	IccTask	delay
DELETE	IccFile	deleteRecord
DELETE	IccFile	deleteLockedRecord
DELETEQ TD	IccDataQueue	empty
DELETEQ TS	IccTempStore	empty
DEQ	IccSemaphore	unlock
DUMP TRANSACTION	IccTask	dump
DUMP TRANSACTION	IccTask	setDumpOpts
ENDBR	IccFileIterator	IccFileIterator (destructor)
ENQ	IccSemaphore	lock
ENQ	IccSemaphore	tryLock
ENTER TRACENUM	IccTask	enterTrace
EXTRACT ATTRIBUTES	IccSession	state, stateText
EXTRACT PROCESS	IccSession	extractProcess
FORMATTIME YYDDD, YYMMDD, etc	IccClock	date
FORMATTIME DATE	IccClock	date
FORMATTIME DATEFORM	IccSystem	dateFormat
FORMATTIME DAYCOUNT	IccClock	daysSince1900
FORMATTIME DAYOFWEEK	IccClock	dayOfWeek
FORMATTIME DAYOFMONTH	IccClock	dayOfMonth
FORMATTIME MONTHOFYEAR	IccClock	monthOfYear
FORMATTIME TIME	IccClock	time
FORMATTIME YEAR	IccClock	year
FREE	IccSession	free
FREEMAIN	IccTask	freeStorage
GETMAIN	IccTask	getStorage
HANDLE ABEND	IccControl	setAbendHandler, cancelAbendHandler, resetAbendHandler
INQUIRE FILE ACCESSMETHOD	IccFile	accessMethod
INQUIRE FILE ADD	IccFile	isAddable
INQUIRE FILE BROWSE	IccFile	isBrowsable
INQUIRE FILE DELETE	IccFileControl	isDeletable
INQUIRE FILE EMPTYSTATUS	IccFile	isEmptyOn
INQUIRE FILE ENABLESTATUS	IccFile	enableStatus

EXEC CICS	Class	Method
INQUIRE FILE KEYPOSITION	IccFile	keyPosition
INQUIRE FILE OPENSTATUS	IccFile	openStatus
INQUIRE FILE READ	IccFile	isReadable
INQUIRE FILE RECORDFORMAT	IccFile	recordFormat
INQUIRE FILE RECORDSIZE	IccFile	recordLength
INQUIRE FILE RECOVSTATUS	IccFile	isRecoverable
INQUIRE FILE TYPE	IccFile	type
INQUIRE FILE UPDATE	IccFile	isUpdatable
ISSUE ABEND	IccSession	issueAbend
ISSUE CONFIRMATION	IccSession	issueConfirmation
ISSUE ERROR	IccSession	issueError
ISSUE PREPARE	IccSession	issuePrepare
ISSUE SIGNAL	IccSession	issueSignal
LINK	IccProgram	link
LINK INPUTMSG INPUTMSGLEN	IccProgram	setInputMessage
LOAD	IccProgram	load
POST	IccClock	setAlarm
READ	IccFile	readRecord
READNEXT	IccFileIterator	readNextRecord
READPREV	IccFileIterator	readPreviousRecord
READQ TD	IccDataQueue	readItem
READQ TS	IccTempStore	readItem
RECEIVE (APPC)	IccSession	receive
RECEIVE (3270)	IccTerminal	receive, receive3270Data
RELEASE	IccProgram	unload
RESETBR	IccFileIterator	reset
RETRIEVE	IccStartRequestQ	retrieveData [1]

Note: The **retrieveData** method gets the start information from CICS and stores it in the IccStartRequestQ object: the information can then be accessed using **data, queueName, returnTermId and returnTransId** methods.

RETRIEVE INTO, LENGTH	IccStartRequestQ	data
RETRIEVE QUEUE	IccStartRequestQ	queueName
RETRIEVE RTRANSID	IccStartRequestQ	returnTransId
RETRIEVE RTERMID	IccStartRequestQ	returnTermId
RETURN	IccControl	main [2]

EXEC CICS	Class	Method
Note: Returning (using C++ reserved word **return**) from method **run** in class **IccControl** results in an EXEC CICS RETURN.		
RETURN TRANSID	IccTerminal	setNextTransId [3]
RETURN IMMEDIATE	IccTerminal	setNextTransId [3]
RETURN COMMAREA LENGTH	IccTerminal	setNextCommArea [3]
RETURN INPUTMSG, INPUTMSGLEN	IccTerminal	setNextInputMessage [3]
Note: Issue this call before returning from **IccControl::run**.		
REWRITE	IccFile	rewriteRecord
SEND (APPC)	IccSession	send, sendInvite, sendLast
SEND (3270)	IccTerminal	send, sendLine
SEND CONTROL CURSOR	IccTerminal	setCursor setLine, setNewLine
SEND CONTROL ERASE	IccTerminal	erase
SEND CONTROL FREEKB	IccTerminal	freeKeyboard
SET FILE ADD|BROWSE| DELETE|...	IccFile	setAccess
SET FILE EMPTYSTATUS	IccFile	setEmptyOnOpen
SET FILE OPEN STATUS| ENABLESTATUS	IccFile	setStatus
SIGNOFF	IccTerminal	signoff
SIGNON	IccTerminal	signon
START TRANSID AT/AFTER	IccStartRequestQ	start [4]
START TRANSID FROM LENGTH	IccStartRequestQ	setData, registerDataBuffer [4]
START TRANSID NOCHECK	IccStartRequestQ	setStartOpts [4]
START TRANSID PROTECT	IccStartRequestQ	setStartOpts [4]
START TRANSID QUEUE	IccStartRequestQ	setQueueName [4]
START TRANSID REQID	IccStartRequestQ	start [4]
START TRANSID TERMID	IccStartRequestQ	start [4]
START TRANSID USERID	IccStartRequestQ	start [4]
START TRANSID RTERMID	IccStartRequestQ	setReturnTermId [4]
START TRANSID RTRANSID	IccStartRequestQ	setReturnTransId [4]
Note: Use methods **setData, setQueueName, setReturnTermId, setReturnTransId, setStartOpts** to set the state of the **IccStartRequestQ** object before issuing start requests with the **start** method.		
STARTBR	IccFileIterator	IccFileIterator (constructor)
SUSPEND	IccTask	suspend
SYNCPOINT	IccTask	commitUOW

EXEC CICS	Class	Method
SYNCPOINT ROLLBACK	IccTask	rollBackUOW
UNLOCK	IccFile	unlockRecord
VERIFY PASSWORD	IccUser	verifyPassword
WAIT CONVID	IccSession	flush
WAIT EVENT	IccTask	waitOnAlarm
WAIT EXTERNAL	IccTask	waitExternal
WAIT JOURNALNUM	IccJournal	wait
WRITE	IccFile	writeRecord
WRITE OPERATOR	IccConsole	write, writeAndGetReply
WRITEQ TD	IccDataQueue	writeItem
WRITEQ TS	IccTempStore	writeItem, rewriteItem

Mapping Foundation Class methods to EXEC CICS calls

The following table shows the correspondence between CICS calls made using the Foundation Classes and the equivalent EXEC CICS API calls.

Table 1. IccAbendData Class

Method	EXEC CICS
abendCode	ASSIGN ABCODE
ASRAInterrupt	ASSIGN ASRAINTRPT
ASRAKeyType	ASSIGN ASRAKEY
ASRAPSW	ASSIGN ASRAPSW
ASRARegisters	ASSIGN ASRAREGS
ASRASpaceType	ASSIGN ASRASPC
ASRAStorageType	ASSIGN ASRASTG
isDumpAvailable	ASSIGN ABDUMP
originalAbendCode	ASSIGN ORGABCODE
programName	ASSIGN ABPROGRAM

Table 2. IccAbsTime Class

Method	EXEC CICS
date	FORMATTIME YYDDD/YYMMDD/etc.
dayOfMonth	FORMATTIME DAYOFMONTH
dayOfWeek	FORMATTIME DAYOFWEEK
daysSince1900	FORMATTIME DAYCOUNT
monthOfYear	FORMATTIME MONTHOFYEAR
time	FORMATTIME TIME

Table 2. IccAbsTime Class (continued)	
Method	**EXEC CICS**
year	FORMATTIME YEAR

Table 3. IccClock Class	
Method	**EXEC CICS**
cancelAlarm	CANCEL
date	FORMATTIME YYDDD/YYMMDD/etc.
dayOfMonth	FORMATTIME DAYOFMONTH
dayOfWeek	FORMATTIME DAYOFWEEK
daysSince1900	FORMATTIME DAYCOUNT
monthOfYear	FORMATTIME MONTHOFYEAR
setAlarm	POST
time	FORMATTIME TIME
update	ASKTIME
year	FORMATTIME YEAR

Table 4. IccConsole Class	
Method	**EXEC CICS**
write	WRITE OPERATOR
writeAndGetReply	WRITE OPERATOR

Table 5. IccControl Class	
Method	**EXEC CICS**
callingProgramId	ASSIGN INVOKINGPROG
cancelAbendHandler	HANDLE ABEND CANCEL
commArea	ADDRESS COMMAREA
initData	ASSIGN INITPARM & INITPARMLEN
programId	ASSIGN PROGRAM
resetAbendHandler	HANDLE ABEND RESET
setAbendHandler	HANDLE ABEND PROGRAM

Table 6. IccDataQueue Class	
Method	**EXEC CICS**
empty	DELETEQ TD
readItem	READQ TD
writeItem	WRITEQ TD

Table 7. IccFile Class	
Method	**EXEC CICS**
access	INQUIRE FILE ADD|BROWSE|DELETE|READ| UPDATE
accessMethod	INQUIRE FILE ACCESSMETHOD
deleteRecord	DELETE FILE RIDFLD
deleteLockedRecord	DELETE FILE
enableStatus	INQUIRE FILE ENABLESTATUS
isAddable	INQUIRE FILE ADD
isBrowsable	INQUIRE FILE BROWSE
isDeletable	INQUIRE FILE DELETE
isEmptyOnOpen	INQUIRE FILE EMPTYSTATUS
isReadable	INQUIRE FILE READ
isRecoverable	INQUIRE FILE RECOVSTATUS
isUpdatable	INQUIRE FILE UPDATE
keyPosition	INQUIRE FILE KEYPOSITION
openStatus	INQUIRE FILE OPENSTATUS
readRecord	READ FILE
recordFormat	INQUIRE FILE RECORDFORMAT
recordLength	INQUIRE FILE RECORDSIZE
rewriteRecord	REWRITE FILE
setAccess	SET FILE ADD BROWSE DELETE etc.
setEmptyOnOpen	SET FILE EMPTYSTATUS
setStatus	SET FILE OPENSTATUS ENABLESTATUS
type	INQUIRE FILE TYPE
unlockRecord	UNLOCK FILE
writeRecord	WRITE FILE

Table 8. IccFileIterator Class	
Method	**EXEC CICS**
IccFileIterator (constructor)	STARTBR FILE
~IccFileIterator (destructor)	ENDBR FILE
readNextRecord	READNEXT FILE
readPreviousRecord	READPREV FILE
reset	RESETBR FILE

Table 9. IccJournal Class	
Method	**EXEC CICS**
wait	WAIT JOURNALNUM
writeRecord	WRITE JOURNALNUM

Table 10. IccProgram Class	
Method	**EXEC CICS**
link	LINK PROGRAM
load	LOAD PROGRAM
unload	RELEASE PROGRAM

Table 11. IccResource Class	
Method	**EXEC CICS**
condition	(RESP & RESP2)
setRouteOption	(SYSID)

Table 12. IccSemaphore Class	
Method	**EXEC CICS**
lock	ENQ RESOURCE
tryLock	ENQ RESOURCE NOSUSPEND
unlock	DEQ RESOURCE

Table 13. IccSession Class	
Method	**EXEC CICS**
allocate	ALLOCATE
connectProcess	CONNECT PROCESS CONVID
converse	CONVERSE CONVID
extractProcess	EXTRACT PROCESS CONVID
flush	WAIT CONVID
free	FREE CONVID
issueAbend	ISSUE ABEND CONVID
issueConfirmation	ISSUE CONFIRMATION CONVID
issueError	ISSUE ERROR CONVID
issuePrepare	ISSUE PREPARE CONVID
issueSignal	ISSUE SIGNAL CONVID
receive	RECEIVE CONVID
send	SEND CONVID
sendInvite	SEND CONVID INVITE

Table 13. IccSession Class (continued)	
Method	**EXEC CICS**
sendLast	SEND CONVID LAST
state	EXTRACT ATTRIBUTES

Table 14. IccStartRequestQ Class	
Method	**EXEC CICS**
cancel	CANCEL
retrieveData	RETRIEVE
start	START TRANSID

Table 15. IccSystem Class	
Method	**EXEC CICS**
applName	ASSIGN APPLID
beginBrowse	INQUIRE (FILE, TDQUEUE, etc) START
dateFormat	FORMATTIME DATEFORM
endBrowse	INQUIRE (FILE, TDQUEUE, etc) END
freeStorage	FREEMAIN
getFile	INQUIRE FILE
getNextFile	INQUIRE FILE NEXT
getStorage	GETMAIN SHARED
operatingSystem	INQUIRE SYSTEM OPSYS
operatingSystemLevel	INQUIRE SYSTEM OPREL
release	INQUIRE SYSTEM RELEASE
releaseText	INQUIRE SYSTEM RELEASE
sysId	ASSIGN SYSID
workArea	ADDRESS CWA

Table 16. IccTask Class	
Method	**EXEC CICS**
abend	ABEND
commitUOW	SYNCPOINT
delay	DELAY
dump	DUMP TRANSACTION
enterTrace	ENTER TRACENUM
facilityType	ASSIGN STARTCODE, TERMCODE, PRINSYSID, FCI
freeStorage	FREEMAIN
isCommandSecurityOn	ASSIGN CMDSEC

Table 16. IccTask Class (continued)	
Method	**EXEC CICS**
isCommitSupported	ASSIGN STARTCODE
isResourceSecurityOn	ASSIGN RESSEC
isRestarted	ASSIGN RESTART
isStartDataAvailable	ASSIGN STARTCODE
principalSysId	ASSIGN PRINSYSID
priority	ASSIGN TASKPRIORITY
rollBackUOW	SYNCPOINT ROLLBACK
setPrioity	CHANGE TASK PRIORITY
startType	ASSIGN STARTCODE
suspend	SUSPEND
triggerDataQueueId	ASSIGN QNAME
userId	ASSIGN USERID
waitExternal	WAIT EXTERNAL / WAITCICS
waitOnAlarm	WAIT EVENT
workArea	ADDRESS TWA

Table 17. IccTempStore Class	
Method	**EXEC CICS**
empty	DELETEQ TS
readItem	READQ TS ITEM
readNextItem	READQ TS NEXT
rewriteItem	WRITEQ TS ITEM REWRITE
writeItem	WRITEQ TS ITEM

Table 18. IccTerminal Class	
Method	**EXEC CICS**
erase	SEND CONTROL ERASE
freeKeyboard	SEND CONTROL FREEKB
height	ASSIGN SCRNHT
netName	ASSIGN NETNAME
receive	RECEIVE
receive3270Data	RECEIVE BUFFER
send	SEND
sendLine	SEND
setCursor	SEND CONTROL CURSOR

Table 18. IccTerminal Class (continued)	
Method	**EXEC CICS**
setLine	SEND CONTROL CURSOR
setNewLine	SEND CONTROL CURSOR
signoff	SIGNOFF
signon	SIGNON
waitForAID	RECEIVE
width	ASSIGN SCRNWD
workArea	ADDRESS TCTUA

Table 19. IccTerminalData Class	
Method	**EXEC CICS**
alternateHeight	ASSIGN ALTSCRNHT
alternateWidth	ASSIGN ALTSCRNWD
defaultHeight	ASSIGN DEFSCRNHT
defaultWidth	ASSIGN DEFSCRNWD
graphicCharSetId	ASSIGN GCHARS
graphicCharCodeSet	ASSIGN GCODES
isAPLKeyboard	ASSIGN APLKYBD
isAPLText	ASSIGN APLTEXT
isBTrans	ASSIGN BTRANS
isColor	ASSIGN COLOR
isEWA	ASSIGN ESASUPP
isExtended3270	ASSIGN EXTDS
isGoodMorning	ASSIGN GMMI
isHighlight	ASSIGN HILIGHT
isKatakana	ASSIGN KATAKANA
isMSRControl	ASSIGN MSRCONTROL
isFieldOutline	ASSIGN OUTLINE
isPS	ASSIGN PS
isSOSI	ASSIGN SOSI
isTextKeyboard	ASSIGN TEXTKYBD
isTextPrint	ASSIGN TEXTPRINT
isValidation	ASSIGN VALIDATION

Table 20. IccUser Class	
Method	**EXEC CICS**
changePassword	CHANGE PASSWORD
verifyPassword	VERIFY PASSWORD

Icc structure

This structure holds global enumerations and functions for the CICS Foundation Classes. These globals are defined within this structure to avoid name conflicts.

Header file: `ICCGLBEH`

Functions

Functions in Icc structure are as follows.

boolText

Returns the text that represents the boolean value described by the parameters, such as "yes" or "on".

static const char* boolText (Bool *test*,
BoolSet *set* = trueFalse**)**

test
A boolean value, defined in this structure, that has one of two values, chosen from a set of values given by *set*.

set
An enumeration, defined in this structure, that indicates from which pair of values *test* is selected. The default is to use true and false.

catchException

This is the function of last resort, used to intercept **IccException** objects that the application fails to catch. It can be called from the **main** function in the stub program, listed in `ICCMAIN` header file, and described in “main function” on page 258. All OO CICS programs should use this stub or a close equivalent.

static void catchException(IccException&*exception*)

exception
A reference to an **IccException** object that holds information about a particular type of exception.

conditionText

Returns the symbolic name associated with a condition value. For example, if **conditionText** is called with *condition* of IccCondition::NORMAL, it returns "NORMAL", if it is called with *condition* of IccCondition::IOERR, it returns "IOERR", and so on.

static const char* conditionText(IccCondition::Codes *condition*)

condition
An enumeration, defined in the **IccCondition** structure, that indicates the condition returned by a call to CICS.

initializeEnvironment

Initializes the CICS Foundation Classes. The rest of the class library can only be called after this function has been called. It is called from the **main** function in the stub program, listed in `ICCMAIN` header file, and described in dfhal1h.dita. All OO CICS programs should use this stub or a close equivalent.

static void initializeEnvironment (ClassMemoryMgmt *mem* = cmmDefault**,**
FamilySubset *fam* = fsDefault**,**
Icc::Bool *EDF*)

mem
An enumeration, defined in this structure, that indicates the memory management policy for the foundation classes.

fam
An enumeration, defined in this structure, that indicates whether the use of CICS features that are not available on all platforms is permitted.

EDF
A boolean that indicates whether EDF tracing is initially on.

isClassMemoryMgmtOn

Returns a boolean value, defined in this structure, that indicates whether class memory management is on.

static Bool isClassMemoryMgmtOn()

isEDFOn

Returns a Boolean value, defined in this structure, that indicates whether EDF tracing is on at the global level.

static Bool isEDFOn()

See **setEDF** in this structure, **isEDFOn** and **setEDF** in **IccResource** class on “IccResource class” on page 161 and Program debugging.

isFamilySubsetEnforcementOn

Returns a boolean value, defined in this structure, that indicates whether it is permitted to use CICS features that are not available on all platforms.

static Bool isFamilySubsetEnforcementOn()

returnToCICS

This call returns the program flow to CICS.

static void returnToCICS()

It is called by the **main** function in the stub program, listed in `ICCMAIN` header file, and described in “main function” on page 258. All OO CICS programs should use this stub or a close equivalent.

setEDF

Sets EDF tracing on or off at the global level.

static void setEDF(Icc::Bool *onOff* = off**)**

onOff
A boolean, defined in this structure, that indicates whether EDF tracing is enabled. As EDF is more suitable for tracing programs that use EXEC CICS calls than object oriented programs, the default is off.

unknownException

This function is called by the **main** function in `ICCMAIN` header file and is used to intercept unknown exceptions.

static void unknownException()

See “main function” on page 258 and **catchException** in this structure).

Enumerations

References in this section to other CICS platforms, such as CICS(r) for AIX, are included for completeness. There have been Technology Releases of the CICS Foundation Classes on those platforms.

Bool

Three equivalent pairs of boolean values are as follows.

- true, yes, on
- false, no, off

true, yes, and on evaluate to 1, while false, no, and off evaluate to zero. Thus you can code test functions as follows:

```
if (task()->isStartDataAvailable())
{
    //do something
}
```

Note: 'true' and 'false' are compiler keywords in the z/OS 1.2 C/C++ compiler and will not be generated by ICCGLBEH when using this compiler, or any later version.

BoolSet

BoolSet enumerations are as follows.

- trueFalse
- yesNo
- onOff

ClassMemoryMgmt

ClassMemoryMgmt enumerations are as folows.

cmmDefault
The defaults for the different platforms are:

z/OS
cmmNonCICS

UNIX
cmmCICS

cmmNonCICS
The C++ environment performs the memory management required by the program.

In z/OS Language Environment ensures that the storage for CICS tasks is released at the end of the task, or if the task terminates abnormally.

On CICS for AIX dynamic storage release does not occur at normal or abnormal task termination. This means that programs are susceptible to memory leaks.

cmmCICS
The **new** and **delete** operators defined in **IccBase** class map storage allocations to CICS; storage is automatically released at task termination.

FamilySubset

FamilySubset enumerations are as follows.

fsDefault
The defaults for the different platforms are all the same: fsAllowPlatformVariance

fsEnforce
Enforces Family Subset conformance; that is, it disallows use of any CICS features that are not available on all CICS servers (OS/2, AIX, and z/OS).

Note: CICS OS/2 is no longer supported.

fsAllowPlatformVariance
Allows each platform to access all the CICS features available on that platform.

GetOpt

This enumeration is used on a number of methods throughout the classes. It indicates whether the value held internally by the object is to be returned to the caller, or whether it has to be refreshed from CICS first.

object
If the value has been previously retrieved from CICS and stored within the object, return this stored value. Otherwise, get a copy of the value from CICS and store within the object.

CICS
Force the object to retrieve a fresh value from CICS (and store it within the object) even if there is already a value stored within the object from a previous invocation.

Platforms

Indicates on which operating system the program is being run.

Possible values are:

- OS2
- UNIX
- MVS™

IccAbendData class

This is a singleton class used to retrieve diagnostic information from CICS about a program abend.

IccBase
IccResource
IccAbendData

Header file: ICCABDEH

IccAbendData constructor (protected)

IccAbendData constructor in IccAbendData class

Constructor

IccAbendData()

Public methods

These are the public methods in this class.

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method.

abendCode

Returns the current 4-character abend code.

const char* abendCode(Icc::GetOpt *opt* = Icc::object**)**

opt
An enumeration, defined in the **Icc** structure, that indicates whether a value should be refreshed from CICS or whether the existing value should be retained. The possible values are described under the **GetOpt** enumeration in the **Icc** structure in "GetOpt" on page 60.

Conditions

INVREQ

ASRAInterrupt

Returns 8 characters of status word (PSW) interrupt information at the point when the latest abend with a code of ASRA, ASRB, ASRD, or AICA occurred.The field contains binary zeroes if no ASRA or ASRB abend occurred during the execution of the issuing transaction, or if the abend originally occurred in a remote DPL server program.

const char* ASRAInterrupt(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

ASRAKeyType

Returns an enumeration, defined in **IccValue**, that indicates the execution key at the time of the last ASRA, ASRB, AICA, or AEYD abend, if any.

The possible values are:

CICSEXECKEY
The task was executing in CICS-key at the time of the last ASRA, ASRB, AICA, or AEYD abend. Note that all programs execute in CICS key if CICS subsystem storage protection is not active.

USEREXECKEY
The task was executing in user-key at the time of the last ASRA, ASRB, AICA, or AEYD abend. Note that all programs execute in CICS key if CICS subsystem storage protection is not active.

NONCICS
The execution key at the time of the last abend was not one of the CICS keys; that is, not key 8 or key 9.

NOTAPPLIC
There has not been an ASRA, ASRB, AICA, or AEYD abend.

IccValue::CVDA ASRAKeyType(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

ASRAPSW

Returns an 8-character status word (PSW) at the point when the latest abend with a code of ASRA, ASRB, ASRD, or AICA occurred.The field contains nulls if no ASRA, ASRB, ASRD, or AICA abend occurred during the execution of the issuing transaction, or if the abend originally occurred in a remote DPL server.

const char* ASRAPSW(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

ASRARegisters

Returns the contents of general registers 0–15, as a 64-byte data area, at the point when the latest ASRA, ASRB, ASRD, or AICA abend occurred. The contents of the registers are returned in the order 0, 1, ..., 15.Note that nulls are returned if no ASRA, ASRB, ASRD, or AICA abend occurred during the execution of the issuing transaction, or if the abend originally occurred in a remote DPL server program.

const char* ASRARegisters(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

ASRASpaceType

Returns an enumeration, defined in **IccValue** structure, that indicates what type of space, if any, was in control at the time of the last ASRA, ASRB, AICA, or AEYD abend.

Possible values are:

SUBSPACE
The task was executing in either its own subspace or the common subspace at the time of the last ASRA, ASRB, AICA, or AEYD abend.

BASESPACE
The task was executing in the base space at the time of the last ASRA, ASRB, AICA, or AEYD abend. Note that all tasks execute in the base space if transaction isolation is not active.

NOTAPPLIC
There has not been an ASRA, ASRB, AICA, or AEYD abend.

IccValue::CVDA ASRASpaceType(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

ASRAStorageType

Returns an enumeration, defined in **IccValue** structure, that indicates what type of storage, if any, was being addressed at the time of the last ASRA, ASRB, AICA, or AEYD abend.

Possible values are:

CICS
CICS-key storage is being addressed. This can be in one of the CICS dynamic storage areas (CDSA or ECDSA), or in one of the read-only dynamic storage areas (RDSA or ERDSA) if either of the following apply:

- CICS is running with the NOPROTECT option on the RENTPGM system initialization parameter
- storage protection is not active

USER
User-key storage in one of the user dynamic storage areas (RDSA or ERDSA) is being addressed.

READONLY
Read-only storage in one of the read-only dynamic storage areas (RDSA or ERDSA) when CICS is running with the PROTECT option on the RENTPGM system initialization parameter.

NOTAPPLIC
One of:

- No ASRA or AEYD abend has been found for this task.
- The storage affected by an abend is not managed by CICS.
- The ASRA abend is not caused by a 0C4 abend.
- An ASRB or AICA abend has occurred since the last ASRA or AEYD abend.

IccValue::CVDA ASRAStorageType(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

instance

Returns a pointer to the single **IccAbendData** object. If the object does not already exist, it is created by this method.

static IccAbendData* instance()

isDumpAvailable

Returns a boolean, defined in **Icc** structure, that indicates whether a dump has been produced. If it has, use **programName** method to find the name of the failing program of the latest abend.

Icc::Bool isDumpAvailable(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

originalAbendCode

Returns the original abend code for this task in case of repeated abends.

const char* originalAbendCode(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

programName

Returns the name of the program that caused the abend.

const char* programName(Icc::GetOpt *opt* = Icc::oldValue**)**

Conditions

INVREQ

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase

Method	Class
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccAbsTime class

This class holds information about absolute time, the time in milliseconds that has elapsed since the beginning of the year 1900.

IccBase
  IccResource
    IccTime
      IccAbsTime

Header file: `ICCTIMEH`

IccAbsTime constructor

IccAbsTime constructor in IccAbsTime class.

Constructor (1)

IccAbsTime(const char* *absTime*)

absTime
The 8-byte value of time, in packed decimal format.

Constructor (2)

The copy constructor.

IccAbsTime(const IccAbsTime& *time*)

Public methods

These are the public methods in this class.

date

Returns the date, as a character string.

const char* date (IccClock::DateFormat *format* = IccClock::defaultFormat**,**
char *dateSeparator* = '\0')

format
An enumeration, defined in **IccClock** class, that indicates the format of the date. The default is to use the installation default, the value set when the CICS region is initialized.

dateSeparator
The character that separates the different fields of the date The default is no separation character.

Conditions

INVREQ

dayOfMonth

Returns the day of the month in the range 1 to 31.

unsigned long dayOfMonth()

Conditions

INVREQ

dayOfWeek

Returns an enumeration, defined in **IccClock** class, that indicates the day of the week.

IccClock::DayOfWeek dayOfWeek()

Conditions

INVREQ

daysSince1900

Returns the number of days that have elapsed since the first day of 1900.

unsigned long daysSince1900()

Conditions

INVREQ

hours

Returns the hours component of the time.

virtual unsigned long hours() const

milliSeconds

Returns the number of milliseconds that have elapsed since the first day of 1900.

long double milliSeconds()

minutes

Returns the minutes component of the time.

virtual unsigned long minutes() const

monthOfYear

Returns an enumeration, defined in **IccClock** class, that indicates the month of the year.

IccClock::MonthOfYear monthOfYear()

Conditions

INVREQ

operator=

Assigns one **IccAbsTime** object to another.

IccAbsTime& operator=(const IccAbsTime& *absTime*)

packedDecimal

Returns the time as an 8-byte packed decimal string that expresses the number of milliseconds that have elapsed since the beginning of the year 1900.

const char* packedDecimal() const

seconds

Returns the seconds component of the time.

virtual unsigned long seconds() const

time

Returns the time as a text string.

const char* time(char *timeSeparator* = '\0')

timeSeparator
The character that delimits the time fields. The default is no time separation character.

Conditions

INVREQ

timeInHours

Returns the number of hours that have elapsed since the day began.

unsigned long timeInHours()

timeInMinutes

Returns the number of minutes that have elapsed since the day began.

unsigned long timeInMinutes()

timeInSeconds

Returns the number of seconds that have elapsed since the day began.

unsigned long timeInSeconds()

year

Returns the year as a 4-digit integer, e.g. 1996.

unsigned long year()

Conditions

INVREQ

Inherited public methods

These are the inherited public methods in IccAbsTime class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
hours	IccTime
isEDFOn	IccResource
minutes	IccTime
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
timeInHours	IccTime
timeInMinutes	IccTime
timeInSeconds	IccTime
type	IccTime

Inherited protected methods

Inherited protected methods in IccAbsTime class:

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccAlarmRequestId class

An **IccAlarmRequestId** object represents a unique alarm request.

IccBase
  IccResourceId
    IccRequestId
      IccAlarmRequestId

It contains the 8-character name of the request identifier and a pointer to a 4-byte timer event control area. **IccAlarmRequestId** is used by the **setAlarm** method of **IccClock** class when setting an alarm, and the **waitOnAlarm** method of **IccTask** when waiting for an alarm.

Header file: `ICCRIDEH`

IccAlarmRequestId constructors

IccAlarmRequestId constructors IccAlarmRequestId constructors:

Constructor (1)

Creates a new object with no information present.

IccAlarmRequestId()

Constructor (2)

Creates an object with information already set.

IccAlarmRequestId (const char* *nam*,
const void* *timerECA*)

name
The 8-character name of the request.

timerECA
A pointer to a 4-byte timer event control area.

Constructor (3)

The copy constructor.

IccAlarmRequestId(const IccAlarmRequestId& *id*)

id
A reference to an **IccAlarmRequestId** object.

Public methods

These methods are used to copy information into an **IccAlarmRequestId** object.

isExpired

Returns a boolean, defined in **Icc** structure, that indicates whether the alarm has expired.

Icc::Bool isExpired()

operator= (1)

IccAlarmRequestId& operator=(const IccRequestId& *id*)

id
A reference to an **IccRequestId** object.

operator= (2)

IccAlarmRequestId& operator=(const IccAlarmRequestId& *id*)

id
A reference to an **IccAlarmRequestId** object.

operator= (3)

IccAlarmRequestId& operator=(const char* *requestName*)

requestName
The 8-character name of the alarm request.

setTimerECA

void setTimerECA(const void* *timerECA*)

timerECA
A pointer to a 4-byte timer event control area.

timerECA

Returns a pointer to the 4-byte timer event control area.

const void* timerECA() const

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccBase class

IccBase class is the base class from which ***all*** CICS Foundation Classes are derived.

IccBase

(The methods associated with **IccBase** are described here although, in practice, they can only be called on objects of the derived classes).

Header file: `ICCBASEH`

IccBase constructor (protected)

IccBase constructor (protected) in IccBase class

Constructor

IccBase(ClassType *type*)

type
An enumeration that indicates what the subclass type is. For example, for an **IccTempStore** object, the class type is 'cTempStore'.

Public methods

These are the public methods in this class.

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method in“abendCode” on page 60.

classType

Returns an enumeration that indicates what the subclass type is. For example, for an **IccTempStore** object, the class type is 'cTempStore'. The possible values are listed under **ClassType** on page cltype.dita.

ClassType classType() const

className

Returns the name of the class. For example, an **IccTempStore** object returns "IccTempStore".Suppose a class **MyDataQueue** inherits from **IccDataQueue**. If **MyDataQueue** calls **setClassName("MyDataQueue")**, **MyDataQueue::className(IccBase::customName)** returns "MyDataQueue" and **MyDataQueue::className(IccBase::baseName)** returns "IccDataQueue". An **IccDataQueue** object returns "IccDataQueue" for both *opt* values.

const char* className(NameOpt *opt*=customName)

opt
An enumerator, defined in this class, that indicates whether to return the base name of the class or the name as customized by a derived class.

customClassNum

Returns the number that an application designer has associated with a subclass that he or she has designed.

unsigned short customClassNum() const

operator delete

Destroys an object in an orderly manner.

void operator delete(void* *object*)

object
A pointer to an object that is to be destroyed.

operator new

Creates a new object of given size. This operator enables the Foundation Classes to use CICS storage allocation (see “initializeEnvironment” on page 57).

void* operator new(size_t *size*)

size
The size of the object that is to be created, in bytes.

Protected methods

setClassName

Sets the name of the class. It is useful for diagnostic purposes to be able to get a string representation of the name of the class to which an object belongs.

void setClassName(const char* *className*)

className
The name of the class. For example, if you create a class **MyTempStore** that is a specialization of **IccTempStore**, you might call **setClassName("MyTempStore")**.

setCustomClassNum

Assigns an identification number to a subclass that is not an original part of the classes, as supplied.

void setCustomClassNum(unsigned short *number*)

number

The number that an application designer associates with a subclass for identification purposes.

Enumerations

Enumerations in IccBase class:

ClassType

The names are derived by deleting the first two characters from the name of the class.

The possible values are:

- cAbendData
- cAlarmRequestId
- cBuf
- cClock
- cConsole
- cControl
- cConvId
- cCUSTOM
- cDataQueue
- cDataQueueId
- cEvent
- cException
- cFile
- cFileId
- cFileIterator
- cGroupId
- cJournal
- cJournalId
- cJournalTypeId
- cLockId
- cMessage
- cPartnerId
- cProgram
- cProgramId
- cRecordIndex
- cRequestId
- cSemaphore
- cSession
- cStartRequestQ
- cSysId
- cSystem
- cTask
- cTempStore
- cTempStoreId
- cTermId
- cTerminal

- cTerminalData
- cTime
- cTPNameId
- cTransId
- cUser
- cUserId

Note: cCUSTOM allows the class library to be extended by non-IBM developers.

NameOpt

NameOpt in Enumerations:

See"className" on page 73.

baseName
Returns the default name assigned to the class as provided by IBM.

customName
Returns the name assigned using **setClassName** method from a subclass *or*, if **setClassName** has not been invoked, the same as *baseName*.

IccBuf class

IccBuf class is supplied for the general manipulation of buffers.

IccBase
  IccBuf

This class is used by other classes that make calls to CICS, but does not itself call CICS services. See Buffer objects.

Header file: `ICCBUFEH`

Sample: `ICC$BUF`

IccBuf constructors

IccBuf constructors in IccBuf class:

Constructor (1)

Creates an **IccBuf** object, allocating its own data area with the given length and with all the bytes within it set to NULL.

IccBuf (unsigned long *length* = 0,
  DataAreaType *type* = extensible**)**

length
The initial length of the data area, in bytes. The default length is 0.

type
An enumeration that indicates whether the data area can be dynamically extended. Possible values are extensible or fixed. The default is extensible.

Constructor (2)

Creates an **IccBuf** object that cannot be extended, adopting the given data area as its own.See warning about Internal/External ownership of buffers.

IccBuf (unsigned long *length*,
void* *dataArea*)

length
The length of the supplied data area, in bytes

dataArea
The address of the first byte of the supplied data area.

Constructor (3)

Creates an **IccBuf** object, allocating its own data area with the same length as the *text* string, and copies the string into its data area.

IccBuf (const char* *text*,
DataAreaType *type* = extensible**)**

text
A null-terminated string to be copied into the new **IccBuf** object.

type
An enumeration that indicates whether the data area can be extended. Possible values are **extensible** or **fixed**. The default is **extensible**.

Constructor (4)

The copy constructor—creates a new **IccBuf** object that is a copy of the given object. The created **IccBuf** object ***always*** has an internal data area.

IccBuf(const IccBuf& *buffer*)

buffer
A reference to an **IccBuf** object that is to be copied into the new object.

Public methods

These are the public methods in this class.

append (1)

Appends data from the given data area to the data area in the object.

IccBuf& append (unsigned long *length*,
const void* *dataArea*)

length
The length of the source data area, in bytes

dataArea
The address of the source data area.

append (2)

Append data, in the form of format string and variable argument, to the data area in the object. This is the same as the form used by **printf** in the standard C library. Note that it is the responsibility of the application programmer to ensure that the optional parameters are consistent with the format string.

IccBuf& append (const char* *format*,
...)

format
The null-terminated format string

...
The optional parameters.

assign (1)

Assigns data from the given data area to the data area in the object.

IccBuf& assign (unsigned long *length*,
const void* *dataArea*)

length
The length of the source data area, in bytes

dataArea
The address of the source data area.

assign (2)

Assigns data, in the form of format string and variable argument, to the data area in the object. This is the same as the form used by **printf** in the standard C library.

IccBuf& assign (const char* *format*,
...)

format
The format string

...
The optional parameters.

cut

Makes the specified cut to the data in the data area and returns a reference to the **IccBuf** object.

IccBuf& cut (unsigned long *length*,
unsigned long *offset* = 0)

length
The number of bytes to be cut from the data area.

offset
The offset into the data area. The default is no offset.

dataArea

Returns the address of data at the given offset into the data area.

const void* dataArea(unsigned long *offset* = 0) const

offset
The offset into the data area. The default is no offset.

dataAreaLength

Returns the length of the data area in bytes.

unsigned long dataAreaLength() const

dataAreaOwner

Returns an enumeration that indicates whether the data area has been allocated by the **IccBuf** constructor or has been supplied from elsewhere.

DataAreaOwner dataAreaOwner() const

The possible values are listed under "DataAreaOwner" on page 86.

dataAreaType

DataAreaType dataAreaType() const

Returns an enumeration that indicates whether the data area can be extended. The possible values are listed under "DataAreaType" on page 86.

dataLength

Returns the length of data in the data area. This cannot be greater than the value returned by **dataAreaLength**

unsigned long dataLength() const

insert

Inserts the given data into the data area at the given offset and returns a reference to the **IccBuf** object.

IccBuf& insert (unsigned long *length*,
const void* *dataArea*,
unsigned long *offset* = 0)

length
The length of the data, in bytes, to be inserted into the **IccBuf** object

dataArea
The start of the source data to be inserted into the **IccBuf** object

offset
The offset in the data area where the data is to be inserted. The default is no offset.

isFMHContained

Icc::Bool isFMHContained() const

Returns a boolean, defined in **Icc** structure, that indicates whether the data area contains FMHs (function management headers).

operator const char*

operator const char*() const

Casts an **IccBuf** object to a null terminated string.

```
IccBuf data("Hello World");
cout « (const char*) data;
```

operator= (1)

Assigns data from another buffer object and returns a reference to the **IccBuf** object.

IccBuf& operator=(const IccBuf& *buffer*)

buffer
A reference to an **IccBuf** object.

operator= (2)

Assigns data from a null-terminated string and returns a reference to the **IccBuf** object. See also the **assign** method.

IccBuf& operator=(const char* *text*)

text
The null-terminated string to be assigned to the **IccBuf** object.

operator+= (1)

Appends data from another buffer object and returns a reference to the **IccBuf** object.

IccBuf& operator+=(const IccBuf& *buffer*)

buffer
A reference to an **IccBuf** object.

operator+= (2)

Appends data from a null-terminated string and returns a reference to the **IccBuf** object. See also the **append** method.

IccBuf& operator+=(const char* *text*)

text
The null-terminated string to be appended to the **IccBuf** object.

operator==

Returns a boolean, defined in **Icc** structure, that indicates whether the data contained in the buffers of the two **IccBuf** objects is the same. It is true if the current lengths of the two data areas are the same and the contents are the same.

Icc::Bool operator==(const IccBuf& *buffer*) const

buffer
A reference to an **IccBuf** object.

operator!=

Returns a boolean, defined in **Icc** structure, that indicates whether the data contained in the buffers of the two **IccBuf** objects is different. It is true if the current lengths of the two data areas are different or if the contents are different.

Icc::Bool operator!=(const IccBuf& *buffer*) const

buffer
A reference to an **IccBuf** object.

operator« (1)

Appends another buffer.

operator«(const IccBuf& *buffer*)

operator« (2)

Appends a string.

operator«(const char* *text*)

operator« (3)

Appends a character.

operator«(char *ch*)

operator« (4)

Appends a character.

operator«(signed char *ch*)

operator« (5)

Appends a character.

operator«(unsigned char *ch*)

operator« (6)

Appends a string.

operator«(const signed char* *text*)

operator« (7)

Appends a string.

operator«(const unsigned char* *text*)

operator« (8)

Appends a short.

operator«(short *num*)

operator« (9)

Appends an unsigned short.

operator«(unsigned short *num*)

operator« (10)

Appends a long.

operator«(long *num*)

operator« (11)

Appends an unsigned long.

operator«(unsigned long *num*)

operator« (12)

Appends an integer.

operator«(int *num*)

operator« (13)

Appends a float.

operator«(float *num*)

operator« (14)

Appends a double.

operator«(double *num*)

operator« (15)

Appends a long double.

operator«(long double *num*)

Appends data of various types to the **IccBuf** object. The types are converted to a 'readable' format, for example from a long to a string representation.

overlay

Makes the data area external and fixed. Any existing internal data area is destroyed.See warning about Internal/External ownership of buffers.

IccBuf& overlay (unsigned long *length*,
void* *dataArea*)

length
The length of the existing data area.

dataArea
The address of the existing data area.

replace

Replaces the current contents of the data area at the given offset with the data provided and returns a reference to the **IccBuf** object.

IccBuf& replace (unsigned long *length*,
const void* *dataArea*,
unsigned long *offset* = 0)

length
The length of the source data area, in bytes.

dataArea
The address of the start of the source data area.

offset
The position where the new data is to be written, relative to the start of the **IccBuf** data area. The default is no offset.

setDataLength

Changes the current length of the data area and returns the new length. If the **IccBuf** object is not extensible, the data area length is set to either the original length of the data area or *length* , whichever is less.

unsigned long setDataLength(unsigned long *length*)

length
The new length of the data area, in bytes

setFMHContained

Allows an application program to indicate that a data area contains function management headers.

void setFMHContained(Icc::Bool *yesNo* = Icc::yes**)**

yesNo
A boolean, defined in **Icc** structure, that indicates whether the data area contains FMHs. The default value is yes.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

DataAreaOwner

Indicates whether the data area of a **IccBuf** object has been allocated outside the object.

Possible values are:

internal
The data area has been allocated by the **IccBuf** constructor.

external
The data area has been allocated externally.

DataAreaType

Indicates whether the data area of a **IccBuf** object can be made longer than its original length.

Possible values are:

extensible
The data area can be automatically extended to accommodate more data.

fixed
The data area cannot grow in size. If you attempt to assign too much data, the data is truncated, and an exception is thrown.

IccClock class

The **IccClock** class controls access to the CICS time and date services.

IccBase
IccResource
IccClock

Header file: ICCCLKEH

Sample: ICC$CLK

IccClock constructor

Constructor

IccClock(UpdateMode *update* = manual**)**

update
An enumeration, defined in this class, that indicates whether the clock is to update its time automatically whenever a time or date service is used, or whether it is to wait until an explicit **update** method call is made. If the time is updated manually, the initial clock time is the time when the **IccClock object** object is created.

Public methods

These are the public methods in this class.

absTime

Returns a reference to an **IccAbsTime** object that contains the absolute time as provided by CICS.

IccAbsTime& absTime()

cancelAlarm

Cancels a previous **setAlarm** request if the alarm time has not yet been reached, that is, the request has not expired.

void cancelAlarm(const IccRequestId* *reqId* = 0)

reqId
An optional pointer to the **IccRequestId** object that holds information on an alarm request.

Conditions

ISCINVREQ, NOTAUTH, NOTFND, SYSIDERR

date

Returns the date as a string.

const char* date (DateFormat *format* = defaultFormat**,**
char *dateSeparator* = '\0')

format
An enumeration, defined in this class, that indicates in which format you want the date to be returned.

dateSeparator
The character that is used to separate different fields in the date. The default is no separation character.

Conditions

INVREQ

dayOfMonth

Returns the day component of the date, in the range 1 to 31.

unsigned long dayOfMonth()

Conditions

INVREQ

dayOfWeek

Returns an enumeration, defined in this class, that indicates the day of the week.

DayOfWeek dayOfWeek()

Conditions

INVREQ

daysSince1900

Returns the number of days that have elapsed since 1st January, 1900.

unsigned long daysSince1900()

Conditions

INVREQ

milliSeconds

Returns the number of milliseconds that have elapsed since 00:00 on 1st January, 1900.

long double milliSeconds()

monthOfYear

MonthOfYear monthOfYear()

Returns an enumeration, defined in this class, that indicates the month of the year.

Conditions

INVREQ

setAlarm

Sets an alarm at the time specified in *time*. It returns a reference to an **IccAlarmRequestId** object that can be used to cancel the alarm—see **cancelAlarm** method.

See also the **"waitOnAlarm" on page 207** method of class **IccTask**.

const IccAlarmRequestId& setAlarm (const IccTime& *time*,
const IccRequestId* *reqId* = 0)

time
A reference to an **IccTime** object that contains time information. As **IccTime** is an abstract class *time* is, in practise, an object of class **IccAbsTime**, **IccTimeOfDay**, or **IccTimeInterval**.

reqId
An optional pointer to an **IccRequestId** object that is used to identify this particular alarm request.

Conditions

EXPIRED, INVREQ

time

Returns the time as a text string.

const char* time(char *timeSeparator* = '\0')

timeSeparator
The character that delimits the time fields. The default is no separation character.

Conditions

INVREQ

update

Updates the clock time and date from CICS. See the **IccClock** constructor.

void update()

year

unsigned long year()

Returns the 4-figure year number, such as 1996.

Conditions

INVREQ

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource

Method	Class
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

DateFormat

- defaultFormat
- DDMMYY
- MMDDYY
- YYDDD
- YYDDMM
- YYMMDD
- DDMMYYYY
- MMDDYYYY
- YYYYDDD
- YYYYDDMM
- YYYYMMDD

DayOfWeek

Indicates the day of the week.

- Sunday
- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday

MonthOfYear

Indicates the month of the year.

- January
- February
- March
- April
- May
- June
- July
- August
- September
- October
- November
- December

UpdateMode

Indicates whether the clock is automatically updated.

manual
The clock initially holds the time at which it was created. It is subsequently updated only when an **update** method call is made.

automatic
The clock is updated to the current CICS time and date whenever any time or date method is called (for example, **daysSince1900**).

IccCondition structure

This structure contains an enumeration of all the CICS condition codes.

Header file: `ICCCNDEH`

Enumerations

Codes

The possible values are:

	Value		Value		Value
0	NORMAL	35	TSIOERR	70	NOTAUTH
1	ERROR	36	MAPFAIL	__	
2	RDATT	37	INVERRTERM	72	SUPPRESSED
3	WRBRK	38	INVMPSZ	__	
4	ICCEOF	39	IGREQID	__	
5	EODS	40	OVERFLOW	75	RESIDERR
6	EOC	41	INVLDC	__	
7	INBFMH	42	NOSTG	__	

	Value		Value		Value
8	ENDINPT	43	JIDERR	__	
9	NONVAL	44	QIDERR	__	
10	NOSTART	45	NOJBUFSP	80	NOSPOOL
11	TERMIDERR	46	DSSTAT	81	TERMERR
12	FILENOTFOUND	47	SELNERR	82	ROLLEDBACK
13	NOTFND	48	FUNCERR	83	END
14	DUPREC	49	UNEXPIN	84	DISABLED
15	DUPKEY	50	NOPASSBKRD	85	ALLOCERR
16	INVREQ	51	NOPASSBKWR	86	STRELERR
17	IOERR	__		87	OPENERR
18	NOSPACE	53	SYSIDERR	88	SPOLBUSY
19	NOTOPEN	54	ISCINVREQ	89	SPOLERR
20	ENDFILE	55	ENQBUSY	90	NODEIDERR
21	ILLOGIC	56	ENVDEFERR	91	TASKIDERR
22	LENGERR	57	IGREQCD	92	TCIDERR
23	QZERO	58	SESSIONERR	93	DSNNOTFOUND
24	SIGNAL	59	SYSBUSY	94	LOADING
25	QBUSY	60	SESSBUSY	95	MODELIDERR
26	ITEMERR	61	NOTALLOC	96	OUTDESCERR
27	PGMIDERR	62	CBIDERR	97	PARTNERIDERR
28	TRANSIDERR	63	INVEXITREQ	98	PROFILEIDERR
29	ENDDATA	64	INVPARTNSET	99	NETNAMEIDERR
30	INVTSREQ	65	INVPARTN	100	LOCKED
31	EXPIRED	66	PARTNFAIL	101	RECORDBUSY
32	RETPAGE	__		102	UOWNOTFOUND
33	RTEFAIL	__		103	UOWLNOTFOUND
34	RTESOME	69	USERIDERR		

Range

maxValue
The highest CICS condition, currently 103.

IccConsole class

This is a singleton class that represents the CICS console.

IccBase
IccResource
IccConsole

Header file: `ICCCONEH`

Sample: `ICC$CON`

IccConsole constructor (protected)

Constructor

No more than one of these objects is permitted in a task. An attempt to create more objects causes an exception to be thrown.

IccConsole()

Public methods

These are the public methods in this class.

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method in"abendCode" on page 60.

instance

Returns a pointer to the single **IccConsole** object that represents the CICS console. If the object does not already exist, it is created by this method.

static IccConsole* instance()

put

Writes the data in *send* to the CICS console. **put** is a synonym for **write**. See Polymorphic Behavior.

virtual void put(const IccBuf& *send*)

send
A reference to an **IccBuf** object that contains the data that is to be written to the console.

replyTimeout

unsigned long replyTimeout() const

Returns the length of the reply timeout in milliseconds.

resetRouteCodes

void resetRouteCodes()

Removes all route codes held in the **IccConsole** object.

setAllRouteCodes

void setAllRouteCodes()

Sets all possible route codes in the **IccConsole** object, that is, 1 through 28.

setReplyTimeout (1)

void setReplyTimeout(IccTimeInterval& *interval*)

interval
A reference to a **IccTimeInterval** object that describes the length of the time interval required.

setReplyTimeout (2)

The two different forms of this method are used to set the length of the reply timeout.

void setReplyTimeout(unsigned long *seconds*)

seconds
The length of the time interval required, in seconds.

setRouteCodes

Saves route codes in the object for use on subsequent **write** and **writeAndGetReply** calls. Up to 28 codes can be held in this way.

void setRouteCodes (unsigned short *numRoutes*,
...)

numRoutes
The number of route codes provided in this call—the number of arguments that follow this one.

...
One or more arguments, the number of which is given by *numRoutes*. Each argument is a route code, of type **unsigned short**, in the range 1 to 28.

write

Writes the data in *send* to the CICS console.

void write (const IccBuf& *send*,
SeverityOpt *opt* = none**)**

send
A reference to an **IccBuf** object that contains the data that is to be written to the console.

opt
An enumeration that indicates the severity of the console message.

Conditions

INVREQ, LENGERR, EXPIRED

writeAndGetReply

Writes the data in *send* to the CICS console and returns a reference to an **IccBuf** object that contains the reply from the CICS operator.

const IccBuf& writeAndGetReply (const IccBuf& *send*,
SeverityOpt *opt*= none**)**

send
A reference to an **IccBuf** object that contains the data that is to be written to the console.

opt
An enumeration that indicates the severity of the console message.

Conditions

INVREQ, LENGERR, EXPIRED

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource

Method	Class
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

SeverityOpt

Possible values are:

- none
- warning
- error
- severe

IccControl class

IccControl class controls an application program that uses the supplied Foundation Classes.

IccBase
  IccResource
    IccControl

This class is a singleton class in the application program; each program running under a CICS task has a single **IccControl** object.

IccControl has a pure virtual **run** method, where application code is written, and is therefore an abstract base class. The application programmer must subclass **IccControl**, and implement the **run** method.

Header file: `ICCCTLEH`

IccControl constructor (protected)

Constructor

IccControl()

Public methods

These are the public methods in this class.

callingProgramId

Returns a reference to an **IccProgramId** object that represents the program that called this program. The returned **IccProgramId** reference contains a null name if the executing program was not called by another program.

const IccProgramId& callingProgramId()

Conditions

INVREQ

cancelAbendHandler

Cancels a previously established exit at this logical program level.

void cancelAbendHandler()

Conditions

NOTAUTH, PGMIDERR

commArea

Returns a reference to an **IccBuf** object that encapsulates the COMMAREA—the communications area of CICS memory that is used for passing data between CICS programs and transactions.

IccBuf& commArea()

Conditions

INVREQ

console

Returns a pointer to the single **IccConsole** object. If this object has not yet been created, this method creates the object before returning a pointer to it.

IccConsole* console()

initData

const IccBuf& initData()

Returns a reference to an **IccBuf** object that contains the initialization parameters specified for the program in the INITPARM system initialization parameter.

Conditions

INVREQ

instance

Returns a pointer to the single **IccControl** object. The object is created if it does not already exist.

static IccControl* instance()

isCreated

static Icc::Bool isCreated()

Returns a boolean value that indicates whether the **IccControl** object already exists. Possible values are true or false.

programId

const IccProgramId& programId()

Returns a reference to an **IccProgramId** object that refers to this executing program.

Conditions

INVREQ

resetAbendHandler

Reactivates a previously cancelled abend handler for this logical program level. (See **cancelAbendHandler on page** “cancelAbendHandler” on page 98).

void resetAbendHandler()

Conditions

NOTAUTH, PGMIDERR

returnProgramId

Returns a reference to an **IccProgramId** object that refers to the program that resumes control when this logical program level issues a return.

const IccProgramId& returnProgramId()

run

virtual void run() = 0

This method should be implemented in a subclass of **IccControl** by the application programmer.

session

IccSession* session()

Returns a pointer to the **IccSession** object that represents the principal facility for this program. An exception is thrown if this program does not have a session as its principal facility.

setAbendHandler (1)

void setAbendHandler(const IccProgramId& *programId*)

programId
A reference to the **IccProgramId** object that indicates which program is affected.

setAbendHandler (2)

These methods set the abend handler to the named program for this logical program level.

void setAbendHandler(const char* *programName*)

programName
The name of the program affected.

Conditions

NOTAUTH, PGMIDERR

startRequestQ

Returns a pointer to the **IccStartRequestQ** object. If this object has not yet been created, this method creates the object before returning a pointer to it.

IccStartRequestQ* startRequestQ()

system

IccSystem* system()

Returns a pointer to the **IccSystem** object. If this object has not yet been created, this method creates the object before returning a pointer to it.

task

IccTask* task()

Returns a pointer to the **IccTask** object. If this object has not yet been created, this method creates the object before returning a pointer to it.

terminal

IccTerminal* terminal()

Returns a pointer to the **IccTerminal** object. If this object has not yet been created, this method creates the object before returning a pointer to it.

This method has a condition, that the transaction must have a terminal as its principal facility. That is, there must be a physical terminal involved.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase

Method	Class
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccConvId class

IccConvId class is used to identify an APPC conversation.

IccBase
  IccResourceId
    IccConvId

IccConvId class is used to identify an APPC conversation.

Header file: `ICCRIDEH`

IccConvId constructors

Constructor (1)

IccConvId(const char* *convName*)

convName
The 4-character name of the conversation.

Constructor (2)

The copy constructor.

IccConvId(const IccConvId& *convId*)

convId
A reference to an **IccConvId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccConvId& operator=(const char* *convName*)

operator= (2)

Assigns new value.

IccConvId& operator=(const IccConvId *id*)

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccDataQueue class

This class represents a CICS transient data queue.

IccBase
  IccResource
    IccDataQueue

Header file: ICCDATEH

Sample: ICC$DAT

IccDataQueue constructors

Constructor (1)

IccDataQueue(const IccDataQueueId& *id*)

id
A reference to an **IccDataQueueId** object that contains the name of the CICS transient data queue.

Constructor (2)

IccDataQueue(const char* *queueName*)

queueName
The 4-byte name of the queue that is to be created. An exception is thrown if *queueName* is not valid.

Public methods

These are the public methods in this class.

clear

A synonym for **empty**. See Polymorphic Behavior.

virtual void clear()

empty

void empty()

Empties the queue, that is, deletes all items on the queue.

Conditions

ISCINVREQ, NOTAUTH, QIDERR, SYSIDERR, DISABLED, INVREQ

get

A synonym for **readItem**. See Polymorphic Behavior.

virtual const IccBuf& get()

put

A synonym for **writeItem**. See Polymorphic Behavior.

virtual void put(const IccBuf& *buffer*)

buffer
A reference to an **IccBuf** object that contains data to be put into the queue.

readItem

const IccBuf& readItem()

Returns a reference to an **IccBuf** object that contains one item read from the data queue.

Conditions

IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTOPEN, QBUSY, QIDERR, QZERO, SYSIDERR, DISABLED, INVREQ

writeItem (1)

void writeItem(const IccBuf& *item*)

item
A reference to an **IccBuf** object that contains data to be written to the queue.

writeItem (2)

Writes an item of data to the queue.

void writeItem(const char* *text*)

text
Text that is to be written to the queue.

Conditions

IOERR, ISCINVREQ, LENGERR, NOSPACE, NOTAUTH, NOTOPEN, QIDERR, SYSIDERR, DISABLED, INVREQ

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccDataQueueId class

IccDataQueueId is used to identify a CICS Transient Data Queue name.

IccBase
  IccResourceId
    IccDataQueueId

IccDataQueueId is used to identify a CICS Transient Data Queue name.

Header file: `ICCRIDEH`

IccDataQueueId constructors

Constructor (1)

IccDataQueueId(const char* *queueName*)

queueName
The 4-character name of the queue

Constructor (2)

IccDataQueueId(const IccDataQueueId& *id*)

id
A reference to an **IccDataQueueId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccDataQueueId& operator=(const char* *queueName*)

queueName
The 4-character name of the queue

operator= (2)

Assigns new value.

IccDataQueueId& operator=(const IccDataQueueId& *id*)

id
A reference to an **IccDataQueueId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccEvent class

The **IccEvent** class contains information on a specific CICS call, called a CICS event.

IccBase
  IccEvent

Header file: ICCEVTEH

Sample: ICC$RES1

IccEvent constructor

Constructor

IccEvent (const IccResource* *object*,
const char* *methodName*)

object
A pointer to the **IccResource** object that is responsible for this event.

methodName
The name of the method that caused the event to be created.

Public methods

These are the public methods in this class.

className

Returns the name of the class responsible for this event.

const char* className() const

classType

IccBase::ClassType classType() const

Returns an enumeration, described under **classType** on page “classType” on page 73 in **IccBase** class, that indicates the type of class that is responsible for this event.

condition

Returns an enumerated type that indicates the condition returned from this CICS event. The possible values are described under the **Codes** type in the **IccCondition** structure.

IccCondition::Codes condition(IccResource::ConditionType *type* =
IccResource::majorCode**) const**

type
An enumeration that indicates whether a major code or minor code is being requested. Possible values are 'majorCode' or 'minorCode'. 'majorCode' is the default value.

conditionText

const char* conditionText() const

Returns the text of the CICS condition code, such as "NORMAL" or "LENGERR".

methodName

const char* methodName() const

Returns the name of the method responsible for this event.

summary

const char* summary()

Returns a summary of the CICS event in the form:

```
CICS event summary: IccDataQueue::readItem condition=23 (QZERO) minor=0
```

Inherited public methods

These are the public methods inherited by this class.

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccException class

IccException class contains information about CICS Foundation Class exceptions.

IccBase
  IccException

It is used to create objects that are 'thrown' to application programs. They are generally used for error conditions such as invalid method calls, but the application programmer can also request an exception is thrown when CICS raises a particular condition.

Header file: ICCEXCEH

Samples: ICC$EXC1, ICC$EXC2, ICC$EXC3

IccException constructor

Constructor

IccException (Type *exceptionType*,
IccBase::ClassType *classType*,
const char* *className*,
const char* *methodName*,
IccMessage* *message*,
IccBase* *object* = 0,
unsigned short *exceptionNum* = 0)

exceptionType
An enumeration, defined in this class, that indicates the type of the exception

classType
An enumeration, defined in this class, that indicates from which type of class the exception was thrown

className
The name of the class from which the exception was thrown

methodName
The name of the method from which the exception was thrown

message
A pointer to the **IccMessage** object that contains information about why the exception was created.

object
A pointer to the object that threw the exception

exceptionNum
The unique exception number.

Note: When the **IccException** object is created it takes ownership of the **IccMessage** given on the constructor. When the **IccException** is deleted, the **IccMessage** object is deleted automatically by the **IccException** destructor. Therefore, do not delete the **IccMessage** object before deleting the **IccException** object.

Public methods

These are the public methods in this class.

className

Returns the name of the class responsible for throwing this exception.

const char* className() const

classType

IccBase::ClassType classType() const

Returns an enumeration, described under **ClassType** in **IccBase** class, that indicates the type of class which threw this exception.

message

IccMessage* message() const

Returns a pointer to an **IccMessage** object that contains information on any message associated with this exception.

methodName

const char* methodName() const

Returns the name of the method responsible for throwing this exception.

number

unsigned short number() const

Returns the unique exception number.

This is a useful diagnostic for IBM service. The number uniquely identifies from where in the source code the exception was thrown.

summary

const char* summary()

Returns a string containing a summary of the exception. This combines the **className**, **methodName**, **number**, **Type**, and **IccMessage::text** methods into the following form:

```
CICS exception summary: 094 IccTempStore::readNextItem type=CICSCondition
```

type

Type type() const

Returns an enumeration, defined in this class, that indicates the type of exception.

typeText

const char* typeText() const

Returns a string representation of the exception type, for example, "objectCreationError", "invalidArgument".

Inherited public methods

These are the public methods inherited by this class.

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

Type

objectCreationError
An attempt to create an object was invalid. This happens, for example, if an attempt is made to create a second instance of a singleton class, such as **IccTask**.

invalidArgument
A method was called with an invalid argument. This happens, for example, if an **IccBuf** object with too much data is passed to the **writeItem** method of the **IccTempStore** class by the application program. An attempt to create an **IccFileId** object with a 9-character filename also generates an exception of this type.

invalidMethodCall
A method call cannot proceed. A typical reason is that the object cannot honor the call in its current state. For example, a **readRecord** call on an **IccFile** object is only honored if an **IccRecordIndex** object, to specify ***which*** record is to be read, has already been associated with the file.

CICSCondition
A CICS condition, listed in the **IccCondition** structure, has occurred in the object and the object was configured to throw an exception.

platformError
An operation is invalid because of limitations of this particular platform.

A platformError exception can occur at 3 levels:

1. An object is not supported on this platform.
2. An object is supported on this platform, but a particular method is not.
3. A method is supported on this platform, but a particular positional parameter is not.

See Platform differences for more details.

familyConformanceError
Family subset enforcement is on for this program and an operation that is not valid on all supported platforms has been attempted.

internalError
The CICS Foundation Classes have detected an internal error. Please call your support organization.

IccFile class

IccFile class enables the application program to access CICS files.

IccBase
  IccResource
    IccFile

Header file: `ICCFILEH`

Sample: `ICC$FIL`

IccFile constructors

Constructor (1)

IccFile (const IccFileId& *id*,
IccRecordIndex* *index* = 0)

id
A reference to the **IccFileId** object that identifies which file is being operated on

index
An optional pointer to the **IccRecordIndex** object that identifies which record in the file is being operated on.

Constructor (2)

To access files using an **IccFile** object, it must have an **IccRecordIndex** object associated with it. If this association is not made when the object is created, use the **registerRecordIndex** method.

IccFile (const char* *fileName*,
IccRecordIndex* *index* = 0)

fileName
The 8-character name of the file

index
An optional pointer to the **IccRecordIndex** object that identifies which record in the file is being operated on.

Public methods

These are the public methods in this class.

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method in“abendCode” on page 60.

access

Returns a composite number indicating the access properties of the file. See also **isReadable**, **isBrowsable**, **isAddable**, **isDeletable**, and **isUpdatable** methods.

unsigned long access(Icc::GetOpt *opt* =Icc::object)

opt
An enumeration, defined in **Icc** structure, that indicates whether you can use a value previously retrieved from CICS (object), or whether the object should retrieve a fresh value from CICS.

accessMethod

Returns an enumeration, defined in **IccValue**, that represents the access method for this file.

Possible values are:

- VSAM
- BDAM
- SFS

IccValue::CVDA accessMethod(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

beginInsert (VSAM only)

Signals the start of a mass insertion of data into the file.

void beginInsert()

deleteLockedRecord

Deletes a record that has been previously locked by **readRecord** method in update mode. (See also **readRecord** method.)

void deleteLockedRecord(unsigned long *updateToken* = 0)

updateToken
A token that indicates which previously read record is to be deleted. This is the token that is returned from **readRecord** method when in update mode.

Conditions

DISABLED, DUPKEY, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFIND, NOTOPEN, SYSIDERR, LOADING

deleteRecord

Deletes one or more records, as specified by the associated **IccRecordIndex** object, and returns the number of deleted records.

unsigned short deleteRecord()

Conditions

DISABLED, DUPKEY, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFIND, NOTOPEN, SYSIDERR, LOADING

enableStatus

Returns an enumeration, defined in **IccValue**, that indicates whether the file is enabled to be used by programs.

Possible values are:

- DISABLED
- DISABLING
- ENABLED
- UNENABLED

IccValue::CVDA enableStatus(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

endInsert (VSAM only)

Marks the end of a mass insertion operation. See **beginInsert**.

void endInsert()

isAddable

Indicates whether more records can be added to the file.

Icc::Bool isAddable(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isBrowsable

Indicates whether the file can be browsed.

Icc::Bool isBrowsable(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isDeletable

Indicates whether the records in the file can be deleted.

Icc::Bool isDeletable(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isEmptyOnOpen

Returns a Boolean that indicates whether the EMPTYREQ option is specified. EMPTYREQ causes the object associated with this file to be set to empty when opened, if it is a VSAM data set defined as reusable.

Icc::Bool isEmptyOnOpen(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isReadable

Indicates whether the file records can be read.

Icc::Bool isReadable(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isRecoverable

Icc::Bool isRecoverable(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

isUpdatable

Indicates whether the file can be updated.

Icc::Bool isUpdatable(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

keyLength

Returns the length of the search key.

unsigned long keyLength(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

keyPosition

Returns the position of the key field in each record relative to the beginning of the record. If there is no key, zero is returned.

long keyPosition(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

openStatus

Returns a CVDA that indicates the open status of the file. Possible values are:

IccValue::CVDA openStatus(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

CLOSED
The file is closed.

CLOSING
The file is in the process of being closed. Closing a file may require dynamic deallocation of data sets and deletion of shared resources, so the process may last a significant length of time.

CLOSEREQUEST
The file is open and one or more application tasks are using it. A request has been received to close it.

OPEN
The file is open.

OPENING
The file is in the process of being opened.

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

readRecord

Reads a record and returns a reference to an **IccBuf** object that contains the data from the record.

const IccBuf& readRecord (ReadMode *mode* = normal**,**
unsigned long* *updateToken* = 0)

mode
An enumeration, defined in this class, that indicates in which mode the record is to be read.

updateToken
A pointer to an **unsigned long** token that will be updated by the method when *mode* is update and you want to make multiple read updates. The token uniquely identifies the update request and is passed to the **deleteLockedRecord**, **rewriteRecord**, or **unlockRecord** methods

Conditions

DISABLED, DUPKEY, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR, LOADING

recordFormat

Returns a CVDA that indicates the format of the data. Possible values are:

IccValue::CVDA recordFormat(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

FIXED
The records are of fixed length.

UNDEFINED (BDAM data sets only)
The format of records on the file is undefined.

VARIABLE
The records are of variable length. If the file is associated with a data table, the record format is always variable length, even if the source data set contains fixed-length records.

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

recordIndex

Returns a pointer to an **IccRecordIndex** object that indicates which records are to be accessed when using methods such as **readRecord**, **writeRecord**, and **deleteRecord**.

IccRecordIndex* recordIndex() const

recordLength

Returns the length of the current record.

unsigned long recordLength(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

registerRecordIndex

void registerRecordIndex(IccRecordIndex* *index*)

index
A pointer to an **IccKey**, **IccRBA**, or **IccRRN** object that will be used by methods such as **readRecord**, **writeRecord**, etc..

rewriteRecord

Updates a record with the contents of *buffer*.

void rewriteRecord (const IccBuf& *buffer*,
unsigned long *updateToken* = 0)

buffer
A reference to the **IccBuf** object that holds the new record data to be written to the file.

updateToken
The token that identifies which previously read record is to be rewritten. See **readRecord**.

Conditions

DISABLED, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR, LOADING

setAccess

Sets the permitted access to the file.

For example:

```
file.setAccess(IccFile::readable + IccFile::notUpdatable);
```

void setAccess(unsigned long *access*)

access
A positive integer value created by ORing (or adding) one or more of the values of the Access enumeration, defined in this class.

Conditions

FILENOTFOUND, INVREQ, IOERR, NOTAUTH

setEmptyOnOpen

void setEmptyOnOpen(Icc::Bool *trueFalse*)

Specifies whether or not to make the file empty when it is next opened.

Conditions

FILENOTFOUND, INVREQ, IOERR, NOTAUTH

setStatus

Sets the status of the file.

void setStatus(Status *status*)

status
An enumeration, defined in this class, that indicates the required status of the file after this method is called.

Conditions

FILENOTFOUND, INVREQ, IOERR, NOTAUTH

type

Returns a CVDA that identifies the type of data set that corresponds to this file. Possible values are:

IccValue::CVDA type(Icc::GetOpt *opt* = Icc::object**)**

opt
See **access** method.

ESDS
The data set is an entry-sequenced data set.

KEYED
The data set is addressed by physical keys.

KSDS
The data set is a key-sequenced data-set.

NOTKEYED
The data set is not addressed by physical keys.

RRDS
The data set is a relative record data set.

VRRDS
The data set is a variable relative record data set.

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

unlockRecord

Unlock a record, previously locked by reading it in update mode. See **readRecord**.

void unlockRecord(unsigned long *updateToken* = 0)

updateToken
A token that indicates which previous **readRecord** update request is to be unlocked.

Conditions

DISABLED, FILENOTFOUND, ILLOGIC, IOERR, ISCINVREQ, NOTAUTH, NOTOPEN, SYSIDERR, INVREQ

writeRecord

Write either a single record or a sequence of records, if used with the **beginInsert** and **endInsert** methods.

void writeRecord(const IccBuf& *buffer*)

buffer
A reference to the **IccBuf** object that holds the data that is to be written into the record.

Conditions

DISABLED, DUPREC, FILENOTFOUND, ILLOGIC, INVREEQ, IOERR, ISCINVREQ, LENGERR, NOSPACE, NOTAUTH, NOTOPEN, SYSIDERR, LOADING, SUPPRESSED

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource

Method	Class
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

Access

readable
File records can be read by CICS tasks.

notReadable
File records cannot be read by CICS tasks.

browsable
File records can be browsed by CICS tasks.

notBrowsable
File records cannot be browsed by CICS tasks.

addable
Records can be added to the file by CICS tasks.

notAddable
Records cannot be added to the file by CICS tasks.

updatable
Records in the file can be updated by CICS tasks.

notUpdatable
Records in the file cannot be updated by CICS tasks.

deletable
Records in the file can be deleted by CICS tasks.

notDeletable
Records in the file cannot be deleted by CICS tasks.

fullAccess
Equivalent to readable AND browsable AND addable AND updatable AND deletable.

noAccess
Equivalent to notReadable AND notBrowsable AND notAddable AND notUpdatable AND notDeletable.

ReadMode

ReadMode is the mode in which a file is read.

normal
No update is to be performed (that is, read-only mode)

update
The record is to be updated. The record is locked by CICS until:

- it is rewritten using the **rewriteRecord** method *or*
- it is deleted using the **deleteLockedRecord** method *or*
- it is unlocked using the **unlockRecord** method *or*
- the task commits or rolls back its resource updates *or*
- the task is abended.

SearchCriterion

equalToKey
The search only finds an exact match.

gteqToKey
The search finds either an exact match or the next record in search order.

Status

open
File is open, ready for read/write requests by CICS tasks.

closed
File is closed, and is therefore not currently being used by CICS tasks.

enabled
File is enabled for access by CICS tasks.

disabled
File is disabled from access by CICS tasks.

IccFileId class

IccFileId is used to identify a file name in the CICS system.

IccBase
IccResourceId
IccFileId

Header file: `ICCRIDEH`

IccFileId constructors

Constructor (1)

IccFileId(const char* *fileName*)

fileName
The name of the file.

Constructor (2)

IccFileId(const IccFileId& *id*)

id
A reference to an **IccFileId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccFileId& operator=(const char* *fileName*)

fileName
The 8-byte name of the file.

operator= (2)

Assigns new value.

IccFileId& operator=(const IccFileId& *id*)

id
A reference to an **IccFileId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccFileIterator class

This class is used to create **IccFileIterator** objects that can be used to browse through the records of a CICS file, represented by an **IccFile** object.

IccBase
  IccResource
    IccFileIterator

Header file: ICCFLIEH

Sample: ICC$FIL

IccFileIterator constructor

Constructor

The **IccFile** and **IccRecordIndex** object must exist before the **IccFileIterator** is created.

IccFileIterator (IccFile* ***file*****,**
IccRecordIndex* ***index*****,**
IccFile::SearchCriterion ***search*** **=** IccFile::gteqToKey**)**

file
A pointer to the **IccFile** object that is to be browsed

index
A pointer to the **IccRecordIndex** object that is being used to select a record in the file

search
An enumeration, defined in **IccFile**, that indicates the criterion being used to find a search match. The default is gteqToKey.

Conditions

DISABLED, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR, LOADING

Public methods

These are the public methods in this class.

readNextRecord

Read the record that follows the current record.

const IccBuf& readNextRecord (IccFile::ReadMode *mode* = IccFile::normal**,**
unsigned long* *updateToken* = 0**)**

mode
An enumeration, defined in **IccFile** class, that indicates the type of read request

updateToken
A returned token that is used to identify this unique update request on a subsequent **rewriteRecord**, **deleteLockedRecord**, or **unlockRecord** method on the file object.

Conditions

DUPKEY, ENDFILE, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFIND, SYSIDERR

readPreviousRecord

Read the record that precedes the current record.

const IccBuf& readPreviousRecord (IccFile::ReadMode *mode* = IccFile::normal**,**
unsigned long* *updateToken* = 0**)**

mode
An enumeration, defined in **IccFile** class, that indicates the type of read request.

updateToken
See **readNextRecord**.

Conditions

DUPKEY, ENDFILE, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFIND, SYSIDERR

reset

Resets the **IccFileIterator** object to point to the record identified by the **IccRecordIndex** object and the specified search criterion.

void reset (IccRecordIndex* *index*,
IccFile::SearchCriterion *search* = IccFile::gteqToKey**)**

index
A pointer to the **IccRecordIndex** object that is being used to select a record in the file.

search
An enumeration, defined in **IccFile**, that indicates the criterion being used to find a search match. The default is gteqToKey.

Conditions

FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFND, SYSIDERR

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccGroupId class

IccGroupId class is used to identify a CICS group.

IccBase
  IccResourceId
    IccGroupId

IccGroupId class is used to identify a CICS group.

Header file: `ICCRIDEH`

IccGroupId constructors

Constructor (1)

IccGroupId(const char* *groupName*)

groupName
The 8-character name of the group.

Constructor (2)

The copy constructor.

IccGroupId(const IccGroupId& *id*)

id
A reference to an **IccGroupId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccGroupId& operator=(const char* *groupName*)

groupName
The 8-character name of the group.

operator= (2)

Assigns new value.

IccGroupId& operator=(const IccGroupId& *id*)

id
A reference to an **IccGroupId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccJournal class

IccJournal class represents a user or system CICS journal.

IccBase
  IccResource
    IccJournal

Header file: `ICCJRNEH`

Sample: `ICC$JRN`

IccJournal constructors

Constructor (1)

IccJournal (const IccJournalId& *id*,
unsigned long *options* = 0)

id
A reference to an **IccJournalId** object that identifies which journal is being used.

options
An integer, constructed from the **Options** enumeration defined in this class, that affects the behavior of **writeRecord** calls on the **IccJournal** object. The values may be combined by addition or bitwise ORing, for example:

```
IccJournal::startIO | IccJournal::synchronous
```

The default is to use the system default.

Constructor (2)

IccJournal (unsigned short *journalNum*,
unsigned long *options* = 0)

journalNum
The journal number (in the range 1-99)

options
See above.

Public methods

These are the public methods in this class.

clearPrefix

Clears the current prefix as set by **registerPrefix** or **setPrefix**.If the current prefix was set using **registerPrefix**, then the **IccJournal** class only removes its own reference to the prefix. The buffer itself is left unchanged.If the current prefix was set by **setPrefix**, then the **IccJournal**'s copy of the buffer is deleted.

void clearPrefix()

journalTypeId

Returns a reference to an **IccJournalTypeId** object that contains a 2-byte field used to identify the origin of journal records.

const IccJournalTypeId& journalTypeId() const

put

A synonym for **writeRecord**—puts data into the journal. See Polymorphic Behavior for information on polymorphism.

virtual void put(const IccBuf& *buffer*)

buffer
A reference to an **IccBuf** object that holds data to be put into the journal.

registerPrefix

void registerPrefix(const IccBuf* *prefix*)

Stores pointer to prefix object for use when the **writeRecord** method is called on this **IccJournal** object.

setJournalTypeId (1)

void setJournalTypeId(const IccJournalTypeId& *id*)

setJournalTypeId (2)

Sets the journal type—a 2 byte identifier—included in the journal record created when using the **writeRecord** method.

void setJournalTypeId(const char* *jtypeid*)

setPrefix (1)

void setPrefix(const IccBuf& *prefix*)

setPrefix (2)

void setPrefix(const char* *prefix*)

Stores the *current* contents of *prefix* for inclusion in the journal record created when the **writeRecord** method is called.

wait

Waits until a previous journal write has completed.

void wait (unsigned long *requestNum*=0,
unsigned long *option* = 0)

requestNum
The write request. Zero indicates the last write on this journal.

option
An integer that affects the behaviour of **writeRecord** calls on the **IccJournal** object. Values other than 0 should be made from the **Options** enumeration, defined in this class. The values may be combined by addition or bitwise ORing, for example `IccJournal::startIO + IccJournal::synchronous`. The default is to use the system default.

writeRecord (1)

unsigned long writeRecord (const IccBuf& *record*,
unsigned long *option* = 0)

record
A reference to an **IccBuf** object that holds the record

option
See above.

writeRecord (2)

Writes the data in the record to the journal.The returned number represents the particular write request and can be passed to the **wait** method in this class.

unsigned long writeRecord (const char* *record*,
unsigned long *option* = 0)

record
The name of the record

option
See above.

Conditions

IOERR, JIDERR, LENGERR, NOJBUFSP, NOTAUTH, NOTOPEN

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

Options

The behaviour of **writeRecord** calls on the **IccJournal** object.

The values can be combined in an integer by addition or bitwise ORing.

startIO
Specifies that the output of the journal record is to be initiated immediately. If 'synchronous' is specified for a journal that is not frequently used, you should also specify 'startIO' to prevent the requesting task waiting for the journal buffer to be filled. If the journal is used frequently, startIO is unnecessary.

noSuspend
Specifies that the NOJBUFSP condition does not suspend an application program.

synchronous
Specifies that synchronous journal output is required. The requesting task waits until the record has been written.

IccJournalId class

IccJournalId is used to identify a journal number in the CICS sytem.

IccBase
IccResourceId
IccJournalId

Header file: `ICCRIDEH`

IccJournalId constructors

Constructor (1)

IccJournalId(unsigned short *journalNum*)

journalNum
The number of the journal, in the range 1 to 99

Constructor (2)

The copy constructor.

IccJournalId(const IccJournalId& *id*)

id
A reference to an **IccJournalId** object.

Public methods

These are the public methods in this class.

number

Returns the journal number, in the range 1 to 99.

unsigned short number() const

operator= (1)

IccJournalId& operator=(unsigned short *journalNum*)

journalNum
The number of the journal, in the range 1 to 99

operator= (2)

Assigns new value.

IccJournalId& operator=(const IccJournalId& *id*)

id
A reference to an **IccJournalId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccJournalTypeId class

An **IccJournalTypeId** class object is used to help identify the origin of a journal record—it contains a 2-byte field that is included in the journal record.

IccBase
  IccResourceId
    IccJournalTypeId

An **IccJournalTypeId** class object is used to help identify the origin of a journal record—it contains a 2-byte field that is included in the journal record.

Header file: `ICCRIDEH`

IccJournalTypeId constructors

Constructor (1)

IccJournalTypeId(const char* *journalTypeName*)

journalTypeName
A 2-byte identifier used in journal records.

Constructor (2)

IccJournalTypeId(const IccJournalId& *id*)

id
A reference to an **IccJournalTypeId** object.

Public methods

These are the public methods in this class.

operator= (1)

void operator=(const IccJournalTypeId& *id*)

id
A reference to an **IccJournalTypeId** object.

operator= (2)

Sets the 2-byte field that is included in the journal record.

void operator=(const char* *journalTypeName*)

journalTypeName
A 2-byte identifier used in journal records.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccKey class

IccKey class is used to hold a search key for an indexed (KSDS) file.

IccBase
  IccRecordIndex
    IccKey

Header file: ICCRECEH

Sample: ICC$FIL

IccKey constructors

Constructor (1)

IccKey (const char* *initValue*,
Kind *kind* = complete**)**

Constructor (2)

IccKey (unsigned short *completeLength*,
Kind *kind*= complete**)**

Constructor (3)

IccKey(const IccKey& *key*)

Public methods

These are the public methods in this class.

assign

Copies the search key into the **IccKey** object.

void assign (unsigned short *length*,
const void* *dataArea*)

length
The length of the data area

dataArea
A pointer to the start of the data area that holds the search key.

completeLength

Returns the length of the key when it is complete.

unsigned short completeLength() const

kind

Kind kind() const

Returns an enumeration, defined in this class, that indicates whether the key is generic or complete.

operator= (1)

IccKey& operator=(const IccKey& *key*)

operator= (2)

IccKey& operator=(const IccBuf& buffer)

operator= (3)

Assigns new value to key.

IccKey& operator=(const char* *value*)

operator== (1)

Icc::Bool operator==(const IccKey& *key*) const

operator== (2)

Icc::Bool operator==(const IccBuf& *text*) const

operator== (3)

Tests equality.

Icc::Bool operator==(const char* *text*) const

operator!= (1)

Icc::Bool operator !=(const IccKey& *key*) const

operator!= (2)

Icc::Bool operator!=(const IccBuf& *text*) const

operator!= (3)

Tests inequality.

Icc::Bool operator!=(const char* *text*) const

setKind

Changes the type of key from generic to complete or vice versa.

void setKind(Kind *kind*)

kind
An enumeration, defined in this class, that indicates whether the key is generic or complete.

value

const char* value()

Returns the start of the data area containing the search key.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
length	IccRecordIndex
operator delete	IccBase
operator new	IccBase
type	IccRecordIndex
value	IccRecordIndex

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

Kind

complete
Specifies that the supplied key is not generic.

generic
Specifies that the search key is generic. A search is satisfied when a record is found with a key whose prefix matches the supplied key.

IccLockId class

IccLockId class is used to identify a lock request.

IccBase
IccResourceId
IccLockId

IccLockId class is used to identify a lock request.

Header file: `ICCRIDEH`

IccLockId constructors

Constructor (1)

IccLockId(const char* *name*)

name
The 8-character name of the lock request.

Constructor (2)

The copy constructor.

IccLockId(const IccLockId& *id*)

id
A reference to an **IccLockId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccLockId& operator=(const char* *name*)

name
The 8-character name of the lock request.

operator= (2)

Assigns new value.

IccLockId& operator=(const IccLockId& *id*)

id
A reference to an **IccLockId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase

Method	Class
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccMessage class

IccMessage can be used to hold a message description.

IccBase
  IccMessage

It is used primarily by the **IccException** class to describe why the **IccException** object was created.

Header file: ICCMSGEH

IccMessage constructor

Constructor

IccMessage (unsigned short *number*,
const char* *text*,
const char* *className* = 0,
const char* *methodName* = 0)

number
The number associated with the message

text
The text associated with the message

className
The optional name of the class associated with the message

methodName
The optional name of the method associated with the message.

Public methods

These are the public methods in this class.

className

Returns the name of the class with which the message is associated, if any. If there is no name to return, a null pointer is returned.

const char* className() const

methodName

const char* methodName() const

Returns the name of the method with which the message is associated, if any. If there is no name to return, a null pointer is returned.

number

unsigned short number() const

Returns the number of the message.

summary

const char* summary()

Returns the text of the message.

text

const char* text() const

Returns the text of the message in the same way as summary.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase

Method	Class
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccPartnerId class

IccPartnerId class represents CICS remote (APPC) partner transaction definitions.

IccBase
  IccResourceId
    IccPartnerId

IccPartnerId class represents CICS remote (APPC) partner transaction definitions.

Header file: `ICCRIDEH`

IccPartnerId constructors

Constructor (1)

IccPartnerId(const char* *partnerName*)

partnerName
The 8-character name of an APPC partner.

Constructor (2)

The copy constructor.

IccPartnerId(const IccPartnerId& *id*)

id
A reference to an **IccPartnerId** object.

Public methods

operator= (1)

IccPartnerId& operator=(const char* *partnerName*)

partnerName
The 8-character name of an APPC partner.

operator= (2)

Assigns new value.

IccPartnerId& operator=(const IccPartnerId& *id*)

id
A reference to an **IccPartnerId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccProgram class

The **IccProgram** class represents any CICS program outside of your currently executing one, which the **IccControl** object represents.

IccBase
  IccResource
    IccProgram

Header file: `ICCPRGEH`

Sample: `ICC$PRG1`, `ICC$PRG2`, `ICC$PRG3`

IccProgram constructors

Constructor (1)

IccProgram(const IccProgramId& *id*)

id
A reference to an **IccProgramId** object.

Constructor (2)

IccProgram(const char* *progName*)

progName
The 8-character name of the program.

Public methods

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method in“abendCode” on page 60.

address

Returns the address of a program module in memory. This is only valid after a successful **load** call.

const void* address() const

clearInputMessage

Clears the current input message which was set by **setInputMessage** or **registerInputMessage**.If the current input message was set using **registerInputMessage** then only the pointer is deleted: the buffer is left unchanged.If the current input message was set using **setInputMessage** then **clearInputMessage** releases the memory used by that buffer.

void clearInputMessage()

entryPoint

const void* entryPoint() const

Returns a pointer to the entry point of a loaded program module. This is only valid after a successful **load** call.

length

unsigned long length() const

Returns the length of a program module. This is only valid after a successful **load** call.

link

void link (const IccBuf* *commArea* = 0,
const IccTransId* *transId* = 0,
CommitOpt *opt* = noCommitOnReturn**)**

commArea
An optional pointer to the **IccBuf** object that contains the COMMAREA—the buffer used to pass information between the calling program and the program that is being called

transId
An optional pointer to the **IccTransId** object that indicates the name of the mirror transaction under which the program is to run if it is a remote (DPL) program link

opt
An enumeration, defined in this class, that affects the behavior of the link when the program is remote (DPL). The default (noCommitOnReturn) is not to commit resource changes on the remote CICS region until the current task commits its resources. The alternative (commitOnReturn) means that the resources of the remote program are committed whether or not this task subsequently abends or encounters a problem.

Conditions: INVREQ, NOTAUTH, PGMIDERR, SYSIDERR, LENGERR, ROLLEDBACK, TERMERR

Restrictions

Links may be nested, that is, a linked program may **link** to another program. However, due to implementation restrictions, you may only nest such programs 15 times. If this is exceeded, an exception is thrown.

load

void load(LoadOpt *opt* = releaseAtTaskEnd**)**

opt
An enumeration, defined in this class, that indicates whether CICS should automatically allow the program to be unloaded at task termination (releaseAtTaskEnd), or not (hold).

Conditions: NOTAUTH, PGMIDERR, INVREQ, LENGERR

registerInputMessage

Store pointer to InputMessage for when the **link** method is called.

void registerInputMessage(const IccBuf& *msg*)

setInputMessage

Specifies data to be made available, by the **IccSession::receive()** method, to the called program, when using the **link** method in this class.

void setInputMessage(const IccBuf& *msg*)

unload

Allow a program to be unloaded. It can be reloaded by a call to **load**.

void unload()

Conditions

NOTAUTH, PGMIDERR, INVREQ

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

CommitOpt

noCommitOnReturn
Changes to resources on the remote CICS region are not committed until the current task commits its resources. This is the default setting.

commitOnReturn
Changes to resources on the remote CICS region are committed whether or not the current task subsequently abends or encounters a problem.

LoadOpt

releaseAtTaskEnd
Indicates that CICS should automatically allow the program to be unloaded at task termination.

hold
Indicates that CICS should not automatically allow the program to be unloaded at task termination. (In this case, this or another task must explicitly use the **unload** method).

IccProgramId class

IccProgramId objects represent program names in the CICS system.

IccBase
IccResourceId
IccProgramId

Header file: ICCRIDEH

IccProgramId constructors

Constructor (1)

IccProgramId(const char* *progName*)

progName
The 8-character name of the program.

Constructor (2)

The copy constructor.

IccProgramId(const IccProgramId& *id*)

id
A reference to an **IccProgramId** object.

Public methods

operator= (1)

IccProgramId& operator=(const char* *progName*)

progName
The 8-character name of the program.

operator= (2)

Assigns new value.

IccProgramId& operator=(const IccProgramId& *id*)

id
A reference to an **IccProgramId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccRBA class

An **IccRBA** object holds a relative byte address which is used for accessing VSAM ESDS files.

IccBase
  IccRecordIndex
    IccRBA

An **IccRBA** object holds a relative byte address which is used for accessing VSAM ESDS files.

Header file: ICCRECEH

IccRBA constructor

Constructor

IccRBA(unsigned long *initRBA* = 0)

initRBA
An initial value for the relative byte address.

Public methods

operator= (1)

IccRBA& operator=(const IccRBA& *rba*)

operator= (2)

Assigns a new value for the relative byte address.

IccRBA& operator=(unsigned long *num*)

num
A valid relative byte address.

operator== (1)

Icc::Bool operator== (const IccRBA& *rba*) const

operator== (2)

Tests equality

Icc::Bool operator== (unsigned long *num*) const

operator!= (1)

Icc!:Bool operator== (const IccRBA& *rba*) const

operator!= (2)

Tests inequality

Icc::Bool operator!=(unsigned long *num*) const

number

unsigned long number() const

Returns the relative byte address.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
length	IccRecordIndex
operator delete	IccBase
operator new	IccBase
type	IccRecordIndex
value	IccRecordIndex

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccRecordIndex class

CICS File Control Record Identifier.

IccBase
  IccRecordIndex
    IccKey
    IccRBA
    IccRRN

CICS File Control Record Identifier. Used to tell CICS which particular record the program wants to retrieve, delete, or update. **IccRecordIndex** is a base class from which **IccKey**, **IccRBA**, and **IccRRN** are derived.

Header file: ICCRECEH

IccRecordIndex constructor (protected)

Constructor

IccRecordIndex(Type *type*)

type
An enumeration, defined in this class, that indicates whether the index type is key, RBA, or RRN.

Note: This is protected because you should not create **IccRecordIndex** objects; see subclasses **IccKey**, **IccRBA**, and **IccRRN**.

Public methods

length

Returns the length of the record identifier.

unsigned short length() const

type

Type type() const

Returns an enumeration, defined in this class, that indicates whether the index type is key, RBA, or RRN.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

Type

Type indicates the access method.

Possible values are:

- key
- RBA
- RRN

IccRequestId class

An **IccRequestId** is used to hold the name of a request.

IccBase
  IccResourceId
    IccRequestId

An **IccRequestId** is used to hold the name of a request. This request identifier can subsequently be used to cancel a request—see, for example, **start** and **cancel** methods in **IccStartRequestQ** class.

Header file: `ICCRIDEH`

IccRequestId constructors

Constructor (1)

An empty **IccRequestId** object.

IccRequestId()

Constructor (2)

IccRequestId(const char* *requestName*)

requestName
The 8-character name of the request.

Constructor (3)

The copy constructor.

IccRequestId(const IccRequestId& *id*)

id
A reference to an **IccRequestId**.

Public methods

operator= (1)

IccRequestId& operator=(const IccRequestId& *id*)

id
A reference to an **IccRequestId** object whose properties are copied into this object.

operator= (2)

Assigns new value.

IccRequestId& operator=(const char* *reqestName*)

requestName
An 8-character string which is copied into this object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccResource class

IccResource class is a base class that is used to derive other classes.

IccBase
  IccResource

The methods associated with **IccResource** are described here although, in practise, they are only called on objects of derived classes.

IccResource is the parent class for all CICS resources—tasks, files, programs, etc. Every class inherits from **IccBase**, but only those that use CICS services inherit from **IccResource**.

Header file: `ICCRESEH`

Sample: `ICC$RES1`, `ICC$RES2`

IccResource constructor (protected)

Constructor

IccResource(IccBase::ClassType *classType*)

classType
An enumeration that indicates what the subclass type is. For example, for an **IccTempStore** object, the class type is cTempStore. The possible values are listed under **ClassType** in the description of the **IccBase** class.

Public methods

actionOnCondition

Returns an enumeration that indicates what action the class will take in response to the specified condition being raised by CICS. The possible values are described in this class.

ActionOnCondition actionOnCondition(IccCondition::Codes *condition*)

condition
The name of the condition as an enumeration. See **IccCondition** structure for a list of the possible values.

actionOnConditionAsChar

char actionOnConditionAsChar(IccCondition::Codes *condition*)

This method is the same as **actionOnCondition** but returns a character, rather than an enumeration, as follows:

0 (zero)
No action is taken for this CICS condition.

H
The virtual method **handleEvent** is called for this CICS condition.

X
An exception is generated for this CICS condition.

A
This program is abended for this CICS condition.

actionsOnConditionsText

Returns a string of characters, one character for each possible condition. Each character indicates the actions to be performed for that corresponding condition. .

The characters used in the string are described in “actionOnConditionAsChar” on page 162. For example, the string: `0X00H0A ...` shows the actions for the first seven conditions are as follows:

condition 0 (NORMAL)
action=0 (noAction)

condition 1 (ERROR)
action=X (throwException)

condition 2 (RDATT)
action=0 (noAction)

condition 3 (WRBRK)
action=0 (noAction)

condition 4 (ICCEOF)
action=H (callHandleEvent)

condition 5 (EODS)
action=0 (noAction)

condition 6 (EOC)
action=A (abendTask)

const char* actionsOnConditionsText()

clear

Clears the contents of the object. This method is virtual and is implemented, wherever appropriate, in the derived classes. See Polymorphic Behavior for a description of polymorphism. The default implementation in this class throws an exception to indicate that it has not been overridden in a subclass.

virtual void clear()

condition

Returns a number that indicates the condition code for the most recent CICS call made by this object.

unsigned long condition(ConditionType *type* = majorCode**) const**

type
An enumeration, defined in this class, that indicates the type of condition requested. Possible values are majorCode (the default) and minorCode.

conditionText

const char* conditionText() const

Returns the symbolic name of the last CICS condition for this object.

get

virtual const IccBuf& get()

Gets data from the **IccResource** object and returns it as an **IccBuf** reference. This method is virtual and is implemented, wherever appropriate, in the derived classes. See Polymorphic Behavior for a description of polymorphism. The default implementation in this class throws an exception to indicate that it has not been overridden in a subclass.

handleEvent

This virtual function may be re-implemented in a subclass (by the application programmer) to handle CICS events (see **IccEvent** class on page "IccEvent class" on page 108).

virtual HandleEventReturnOpt handleEvent(IccEvent& *event*)

event
A reference to an **IccEvent** object that describes the reason why this method is being called.

id

const IccResourceId* id() const

Returns a pointer to the **IccResourceId** object associated with this **IccResource** object.

isEDFOn

Icc::Bool isEDFOn() const

Returns a boolean value that indicates whether EDF trace is active. Possible values are yes or no.

isRouteOptionOn

Icc::Bool isRouteOptionOn() const

Returns a boolean value that indicates whether the route option is active. Possible values are yes or no.

name

const char* name() const

Returns a character string that gives the name of the resource that is being used. For an **IccTempStore** object, the 8-character name of the temporary storage queue is returned. For an **IccTerminal** object, the 4-character terminal name is returned. This is equivalent to calling **id()→name**.

put

Puts information from the buffer into the **IccResource** object. This method is virtual and is implemented, wherever appropriate, in the derived classes. See Polymorphic Behavior for more information on polymorphism. The default implementation in this class throws an exception to indicate that it has not been overridden in a subclass.

virtual void put(const IccBuf& *buffer*)

buffer
A reference to an **IccBuf** object that contains data that is to be put into the object.

routeOption

const IccSysId& routeOption() const

Returns a reference to an **IccSysId** object that represents the system to which all CICS requests are routed—explicit function shipping.

setActionOnAnyCondition

Specifies the default action to be taken by the CICS foundation classes when a CICS condition occurs.

void setActionOnAnyCondition(ActionOnCondition *action*)

action
The name of the action as an enumeration. The possible values are listed under the description of this class.

setActionOnCondition

Specifies what action is automatically taken by the CICS foundation classes when a given CICS condition occurs.

void setActionOnCondition (ActionOnCondition *action*,
IccCondition::Codes *condition*)

action
The name of the action as an enumeration. The possible values are listed under the description of this class.

condition
See **IccCondition** structure.

setActionsOnConditions

void setActionsOnConditions(const char* *actions* = 0)

actions
A string that indicates what action is to be taken for each condition. The default is not to indicate any actions, in which case each condition is given a default **ActionOnCondition** of noAction. The string should have the same format as the one returned by the **actionsOnConditionsText** method.

setEDF

Switches EDF on or off for this resource object. These methods force the object to route CICS requests to the named remote system. This is called explicit function shipping.

void setEDF(Icc::Bool *onOff*)

onOff
A boolean value that selects whether EDF trace is switched on or off.

setRouteOption (1)

The parameters are:

void setRouteOption(const IccSysId& *sysId*)

sysId
The **IccSysId** object that represents the remote system to which commands are routed.

setRouteOption (2)

This option is only valid for certain classes: Attempting to use this method on other subclasses of **IccResource** causes an exception to be thrown.

Valid classes are:

- **IccDataQueue**
- **IccFile**
- **IccFileIterator**
- **IccProgram**
- **IccStartRequestQ**
- **IccTempStore**

To turn off the route option specify no parameter, for example:

```
obj.setRouteOption()
```

void setRouteOption(const char* *sysName* = 0)

sysName
The 4-character name of the system to which commands are routed.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

ActionOnCondition

Possible values are:

noAction
Carry on as normal; it is the application program's responsibility to test CICS conditions using the **condition** method, after executing a method that calls CICS services.

callHandleEvent
Call the virtual **handleEvent** method.

throwException
An **IccException** object is created and thrown. This is typically used for more serious conditions or errors.

abendTask
Abend the CICS task.

HandleEventReturnOpt

Possible values are:

rContinue
The CICS event proceeded satisfactorily and normal processing is to resume.

rThrowException
The application program could not handle the CICS event and an exception is to be thrown.

rAbendTask
The application program could not handle the CICS event and the CICS task is to be abended.

ConditionType

Possible values are:

majorCode
The returned value is the CICS RESP value. This is one of the values in IccCondition::codes.

minorCode
The returned value is the CICS RESP2 value.

IccResourceId class

This is a base class from which **IccTransId** and other classes, whose names all end in "Id", are derived.

IccBase
  IccResourceId

Many of these derived classes represent CICS resource names.

Header file: `ICCRIDEH`

IccResourceId constructors (protected)

Constructor (1)

IccResourceId (IccBase::ClassType *typ*,
const IccResourceId& *id*)

type
An enumeration, defined in **IccBase** class, that indicates the type of class.

id
A reference to an **IccResourceId** object that is used to create this object.

Constructor (2)

IccResourceId (IccBase::ClassType *type*,
const char* *resName*)

type
An enumeration, defined in **IccBase** class, that indicates the type of class.

resName
The name of a resource that is used to create this object.

Public methods

These are the public methods in this class.

name

Returns the name of the resource identifier as a string. Most **...Id** objects have 4- or 8-character names.

const char* name() const

nameLength

unsigned short nameLength() const

Returns the length of the name returned by the **name** method.

Protected methods

operator=

Set an **IccResourceId** object to be identical to *id*.

IccResourceId& operator=(const IccResourceId& *id*)

id
A reference to an **IccResourceId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccRRN class

An **IccRRN** object holds a relative record number and is used to identify records in VSAM RRDS files.

IccBase
  IccRecordIndex
    IccRRN

An **IccRRN** object holds a relative record number and is used to identify records in VSAM RRDS files.

Header file: `ICCRECEH`

IccRRN constructors

Constructor

IccRRN(unsigned long *initRRN* = 1)

initRRN
The initial relative record number—an integer greater than 0. The default is 1.

Public methods

These are the public methods in this class.

operator= (1)

IccRRN& operator=(const IccRRN& *rrn*)

operator= (2)

Assigns a new value for the relative record number.

IccRRN& operator=(unsigned long *num*)

num
A relative record number—an integer greater than 0.

operator== (1)

Icc::Bool operator== (const IccRRN& *rrn*) const

operator== (2)

Tests equality

Icc::Bool operator== (unsigned long num) const

operator!= (1)

Icc::Bool operator!= (const IccRRN& *rrn*) const

operator!= (2)

Tests inequality

Icc::Bool operator!=(unsigned long num) const

number

unsigned long number() const

Returns the relative record number.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
length	IccRecordIndex
operator delete	IccBase
operator new	IccBase
type	IccRecordIndex
value	IccRecordIndex

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccSemaphore class

This class enables synchronization of resource updates.

IccBase
  IccResource
    IccSemaphore

Header file: ICCSEMEH

Sample: ICC$SEM

IccSemaphore constructor

Constructor (1)

IccSemaphore (const char* *resource*,
LockType *type* = byValue,
LifeTime *life* = UOW)

resource
A text string, if *type* is byValue, otherwise an address in storage.

type
An enumeration, defined in this class, that indicates whether locking is by value or by address. The default is by value.

life
An enumeration, defined in this class, that indicates how long the semaphore lasts. The default is to last for the length of the UOW.

Constructor (2)

IccSemaphore (const IccLockId& *id*,
LifeTime *life* = UOW)

id
A reference to an **IccLockId** object

life
An enumeration, defined in this class, that indicates how long the semaphore lasts. The default is to last for the length of the UOW.

Public methods

These are the public methods in this class.

lifeTime

Returns an enumeration, defined in this class, that indicates whether the lock lasts for the length of the current unit-of-work ('UOW') or until the task terminates('task').

LifeTime lifeTime() const

lock

void lock()

Attempts to get a lock. This method blocks if another task already owns the lock.

Conditions

ENQBUSY, LENGERR, INVREQ

tryLock

Attempts to get a lock. This method does not block if another task already owns the lock. It returns a boolean that indicates whether it succeeded.

Icc::Bool tryLock()

Conditions

ENQBUSY, LENGERR, INVREQ

type

Returns an enumeration, defined in this class, that indicates what type of semaphore this is.

LockType type() const

unlock

void unlock()

Release a lock.

Conditions

LENGERR, INVREQ

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

LockType

byValue
The lock is on the contents (for example, name).

byAddress
The lock is on the memory address.

LifeTime

UOW
The semaphore lasts for the length of the current unit of work.

task
The semaphore lasts for the length of the task.

IccSession class

This class enables APPC and DTP programming.

IccBase
  IccResource
    IccSession

Header file: ICCSESEH

Sample: ICC$SES1, ICC$SES2

IccSession constructors (public)

Constructor (1)

IccSession(const IccPartnerId& *id*)

id
A reference to an **IccPartnerId** object

Constructor (2)

IccSession (const IccSysId& *sysId*,
const char* *profile* = 0)

sysId
A reference to an **IccSysId** object that represents a remote CICS system

profile
The 8-character name of the profile.

Constructor (3)

IccSession (const char* *sysName*,
const char* *profile* = 0)

sysName
The 4-character name of the remote CICS system with which this session is associated

profile
The 8-character name of the profile.

IccSession constructor (protected)

Constructor

This constructor is for back end DTP CICS tasks that have a session as their principal facility. In this case the application program uses the **session** method on the **IccControl** object to gain access to their **IccSession** object.

IccSession()

Public methods

These are the public methods in this class.

allocate

Establishes a session (communication channel) to the remote system.

void allocate(AllocateOpt *option* = queue)

option
An enumeration, defined in this class, that indicates what action CICS is to take if a communication channel is unavailable when this method is called.

Conditions

INVREQ, SYSIDERR, CBIDERR, NETNAMEIDERR, PARTNERIDERR, SYSBUSY

connectProcess (1)

This method can only be used if an **IccPartnerId** object was used to construct this session object.

void connectProcess (SyncLevel *level*,
const IccBuf* *PIP* = 0)

level
An enumeration, defined in this class, that indicates what sync level is to be used for this conversation

PIP
An optional pointer to an **IccBuf** object that contains the PIP data to be sent to the remote system

connectProcess (2)

void connectProcess (SyncLevel *level*,
const IccTransId& *transId*,
const IccBuf* *PIP* = 0)

level
An enumeration, defined in this class, that indicates what sync level is to be used for this conversation

transId
A reference to an **IccTransId** object that holds the name of the transaction to be started on the remote system

PIP
An optional pointer to an **IccBuf** object that contains the PIP data to be sent to the remote system

connectProcess (3)

Starts a partner process on the remote system in preparation for sending and receiving information.

void connectProcess (SyncLevel *level*,
const IccTPNameId& *TPName*,
const IccBuf* *PIP* = 0)

level
An enumeration, defined in this class, that indicates what sync level is to be used for this conversation

TPName
A reference to an **IccTPNameId** object that contains the 1–64 character TP name.

PIP
An optional pointer to an **IccBuf** object that contains the PIP data to be sent to the remote system

Conditions

INVREQ, LENGERR, NOTALLOC, PARTNERIDERR, NOTAUTH, TERMERR, SYSBUSY

converse

converse sends the contents of *send* and returns a reference to an **IccBuf** object that holds the reply from the remote APPC partner.

const IccBuf& converse(const IccBuf& *send*)

send
A reference to an **IccBuf** object that contains the data that is to be sent.

Conditions

EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

convId

Returns a reference to an **IccConvId** object that contains the 4-byte conversation identifier.

const IccConvId& convId()

errorCode

const char* errorCode() const

Returns the 4-byte error code received when **isErrorSet** returns true. See the relevant DTP Guide for more information.

extractProcess

void extractProcess()

Retrieves information from an APPC conversation attach header and holds it inside the object. See **PIPList**, **process**, and **syncLevel** methods to retrieve the information from the object. This method should be used by the back end task if it wants access to the PIP data, the process name, or the synclevel under which it is running.

Conditions

INVREQ, NOTALLOC, LENGERR

flush

Ensure that accumulated data and control information are transmitted on an APPC mapped conversation.

void flush()

Conditions

INVREQ, NOTALLOC

free

Return the APPC session to CICS so that it may be used by other tasks.

void free()

Conditions

INVREQ, NOTALLOC

get

A synonym for **receive**. See Polymorphic Behavior for information on polymorphism.

virtual const IccBuf& get()

isErrorSet

Icc::Bool isErrorSet() const

Returns a boolean variable, defined in **Icc** structure, that indicates whether an error has been set.

isNoDataSet

Icc::Bool isNoDataSet() const

Returns a boolean variable, defined in **Icc** structure, that indicates if no data was returned on a **send**—just control information.

isSignalSet

Icc::Bool isSignalSet() const

Returns a boolean variable, defined in **Icc** structure, that indicates whether a signal has been received from the remote process.

issueAbend

void issueAbend()

Abnormally ends the conversation. The partner transaction sees the TERMERR condition.

Conditions

INVREQ, NOTALLOC, TERMERR

issueConfirmation

Sends positive response to a partner's **send** request that specified the confirmation option.

void issueConfirmation()

Conditions

INVREQ, NOTALLOC, TERMERR, SIGNAL

issueError

Signals an error to the partner process.

void issueError()

Conditions

INVREQ, NOTALLOC, TERMERR, SIGNAL

issuePrepare

This only applies to DTP over APPC links. It enables a syncpoint initiator to prepare a syncpoint worker for syncpointing by sending only the first flow ('prepare to commit') of the syncpoint exchange.

void issuePrepare()

Conditions

INVREQ, NOTALLOC, TERMERR

issueSignal

Signals that a mode change is needed.

void issueSignal()

Conditions

INVREQ, NOTALLOC, TERMERR

PIPList

Returns a reference to an **IccBuf** object that contains the PIP data sent from the front end process. A call to this method should be preceded by a call to **extractProcess** on back end DTP processes.

IccBuf& PIPList()

process

const IccBuf& process() const

Returns a reference to an **IccBuf** object that contains the process data sent from the front end process. A call to this method should be preceded by a call to **extractProcess** on back end DTP processes.

put

A synonym for **send**. See Polymorphic Behavior for information on polymorphism.

virtual void put(const IccBuf& *data*)

data
A reference to an **IccBuf** object that holds the data to be sent to the remote process.

receive

const IccBuf& receive()

Returns a reference to an **IccBuf** object that contains the data received from the remote system.

Conditions

EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

send (1)

void send (const IccBuf& *send*,
SendOpt *option* = normal**)**

send
A reference to an **IccBuf** object that contains the data that is to be sent.

option
An enumeration, defined in this class, that affects the behavior of the **send** method. The default is normal.

send (2)

Sends data to the remote partner.

void send(SendOpt *option* = normal**)**

option
An enumeration, defined in this class, that affects the behavior of the **send** method. The default is normal.

Conditions

INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

sendInvite (1)

void sendInvite (const IccBuf& *send*,
SendOpt *option* = normal**)**

send
A reference to an **IccBuf** object that contains the data that is to be sent.

option
An enumeration, defined in this class, that affects the behavior of the **sendInvite** method. The default is normal.

sendInvite (2)

Sends data to the remote partner and indicates a change of direction, that is, the next method on this object will be **receive**.

void sendInvite(SendOpt *option* = normal**)**

option
An enumeration, defined in this class, that afffects the behavior of the **sendInvite** method. The default is normal.

Conditions

INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

sendLast (1)

void sendLast (const IccBuf& *send*,
SendOpt *option* = normal**)**

send
A reference to an **IccBuf** object that contains the data that is to be sent.

option
An enumeration, defined in this class, that affects the behavior of the **sendLast** method. The default is normal.

sendLast (2)

Sends data to the remote partner and indicates that this is the final transmission. The **free** method must be invoked next, unless the sync level is 2, when you must commit resource updates before the **free**. (See **commitUOW** on page "commitUOW" on page 201 in **IccTaskClass**).

void sendLast(SendOpt *option* = normal**)**

option
An enumeration, defined in this class, that affects the behavior of the **sendLast** method. The default is normal.

Conditions

INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

state

Returns a CVDA, defined in **IccValue** structure, that indicates the current state of the APPC conversation.

Possible values are:

- ALLOCATED
- CONFFREE
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- ROLLBACK
- SEND
- SYNCFREE
- SYNCRECEIVE
- SYNCSEND
- NOTAPPLIC

IccValue::NOTAPPLIC is returned if there is no APPC conversation state.

IccValue::CVDA state(StateOpt *option* = lastCommand**)**

option
An enumeration, defined in this class, that indicates how to report the state of the conversation

Conditions

INVREQ, NOTALLOC

stateText

Returns the symbolic name of the state that **state** method would return. For example, if **state** returns IccValue::ALLOCATED, **stateText** would return "ALLOCATED".

const char* stateText(StateOpt *option* = lastCommand**)**

option
An enumeration, defined in this class, that indicates how to report the state of the conversation

syncLevel

SyncLevel syncLevel() const

Returns an enumeration, defined in this class, that indicates the synchronization level that is being used in this session. A call to this method should be preceded by a call to **extractProcess** on back end DTP processes.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource

Method	Class
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

AllocateOpt

queue
If all available sessions are in use, CICS is to queue this request (and block the method) until it can allocate a session.

noQueue
Control is returned to the application if it cannot allocate a session. CICS raises the SYSBUSY condition.

Indicates whether queuing is required on an **allocate** method.

SendOpt

normal
The default.

confirmation
Indicates that a program using SyncLevel level1 or level2 requires a response from the remote partner program. The remote partner can respond positively, using the **issueConfirmation** method, or negatively, using the **issueError** method. The sending program does not receive control back from CICS until the response is received.

wait
Requests that the data is sent and not buffered internally. CICS is free to buffer requests to improve performance if this option is not specified.

StateOpt

Use StateOpt to indicate how the state of a conversation is to be reported.

lastCommand
Return the state at the time of the completion of the last operation on the session.

extractState
Return the explicitly extracted current state.

SyncLevel

level0
Sync level 0

level1
Sync level 1

level2
Sync level 2

IccStartRequestQ class

This is a singleton class that enables the application programmer to request an asynchronous start of another CICS transaction.

IccBase
IccResource
IccStartRequestQ

(see the **start** method on page "start" on page 190).

An asynchronously started transaction uses the **IccStartRequestQ** class method **retrieveData** to gain the information passed to it by the transaction that issued the **start** request.

An unexpired start request can be cancelled by using the **cancel** method.

Header file: ICCSRQEH

Sample: ICC$SRQ1, ICC$SRQ2

IccStartRequestQ constructor (protected)

Constructor

IccStartRequestQ()

Public methods

These are the public methods in this class.

cancel

Cancels a previously issued **start** request that has not yet expired.

void cancel (const IccRequestId& *reqId*,
const IccTransId* *transId* = 0)

reqId
A reference to an **IccRequestId** object that represents the request to be cancelled

transId
An optional pointer to an **IccTransId** object that represents the transaction that is to be cancelled.

Conditions

ISCINVREQ, NOTAUTH, NOTFND, SYSIDERR

clearData

clearData clears the current data that is to be passed to the started transaction.

void clearData()

The data was set using **setData** or **registerData**.

If the data was set using **registerData**, only the pointer to the data is removed, the data in the buffer is left unchanged.

If the data was set using **setData**, then **clearData** releases the memory used by the buffer.

data

Returns a reference to an **IccBuf** object that contains data passed on a start request. A call to this method should be preceded by a call to **retrieveData** method.

const IccBuf& data() const

instance

static IccStartRequestQ* instance()

Returns a pointer to the single **IccStartRequestQ** object. If the object does not exist it is created. See also **startRequestQ** method on page “startRequestQ” on page 101 of **IccControl**.

queueName

const char* queueName() const

Returns the name of the queue that was passed by the start requester. A call to this method should be preceded by a call to **retrieveData** method.

registerData

Registers an **IccBuf** object to be interrogated for start data on each subsequent **start** method invocation.This just stores the address of the **IccBuf** object within the **IccStartRequestQ** so that the **IccBuf** object can be found when using the **start** method. This differs from the **setData** method, which takes a copy of the data held in the **IccBuf** object during the time that it is invoked.

void registerData(const IccBuf* *buffer*)

buffer
A pointer to the **IccBuf** object that holds data to be passed on a **start** request.

reset

void reset()

Clears any associations previously made by **set...** methods in this class.

retrieveData

Used by a task that was started, via an async start request, to gain access to the information passed by the start requester. The information is returned by the **data**, **queueName**, **returnTermId**, and **returnTransId** methods.

void retrieveData(RetrieveOpt *option* = noWait**)**

option
An enumeration, defined in this class, that indicates what happens if there is no start data available.

Conditions

ENDDATA, ENVDEFERR, IOERR, LENGERR, NOTFND, INVREQ

Note: The ENVDEFERR condition will be raised if all the possible options (**setData**, **setQueueName**, **setReturnTermId**, and **setReturnTransId**) are not used before issuing the **start** method. This condition is therefore not necessarily an error condition and your program should handle it accordingly.

returnTermId

Returns a reference to an **IccTermId** object that identifies which terminal is involved in the session. A call to this method should be preceded by a call to **retrieveData** method.

const IccTermId& returnTermId() const

returnTransId

const IccTransId& returnTransId() const

Returns a reference to an **IccTransId** object passed on a start request. A call to this method should be preceded by a call to **retrieveData** method.

setData

void setData(const IccBuf& *buf*)

Copies the data in *buf* into the **IccStartRequestQ**, which passes it to the started transaction when the **start** method is called. See also **registerData** on page "registerData" on page 187 for an alternative way to pass data to started transactions.

setQueueName

Requests that this queue name be passed to the started transaction when the **start** method is called.

void setQueueName(const char* *queueName*)

queueName
An 8-character queue name.

setReturnTermId (1)

void setReturnTermId(const IccTermId& *termId*)

termId
A reference to an **IccTermId** object that identifies which terminal is involved in the session.

setReturnTermId (2)

Requests that this return terminal ID be passed to the started transaction when the **start** method is called.

void setReturnTermId(const char* *termName*)

termName
The 4-character name of the terminal that is involved in the session.

setReturnTransId (1)

void setReturnTransId(const IccTransId& *transId*)

transId
A reference to an **IccTransId** object.

setReturnTransId (2)

Requests that this return transaction ID be passed to the started transaction when the **start** method is called.

void setReturnTransId(const char* *transName*)

transName
The 4-character name of the return transaction.

setStartOpts

Sets whether the started transaction is to have protection and whether it is to be checked.

void setStartOpts (ProtectOpt *popt* = none**,**
CheckOpt *copt* = check**)**

popt
An enumeration, defined in this class, that indicates whether start requests are to be protected

copt
An enumeration, defined in this class, that indicates whether start requests are to be checked.

start

Asynchronously starts the named CICS transaction. The returned reference to an **IccRequestId** object identifies the **start** request and can be used subsequently to **cancel** the **start** request.

const IccRequestId& start (const IccTransId& *transId*,
const IccTermId* *termId*,
const IccTime* *time* = 0,
const IccRequestId* *reqId* = 0)

or

const IccRequestId& start (const IccTransId& *transId*,
const IccUserId* *userId*,
const IccTime* *time* = 0,
const IccRequestId* *reqId* = 0)

or

const IccRequestId& start (const IccTransId& *transId*,
const IccTime* *time* = 0,
const IccRequestId* *reqId* = 0)

transId
A reference to an **IccTransId** object that represents the transaction to be started

termId
A reference to an **IccTermId** object that identifies which terminal is involved in the session.

userId
A reference to an **IccUserId** object that represents the user ID.

time
An (optional) pointer to an **IccTime** object that specifies when the task is to be started. The default is for the task to be started immediately.

reqId
An (optional) pointer to an **IccRequestId** object that is used to identify this start request so that the **cancel** can cancel the request.

Conditions

INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, SYSIDERR, TERMIDERR, TRANSIDERR, USERIDERR

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

RetrieveOpt

- noWait
- wait

ProtectOpt

- none
- protect

CheckOpt

- check
- noCheck

IccSysId class

IccSysId class is used to identify a remote CICS system.

IccBase
  IccResourceId
    IccSysId

IccSysId class is used to identify a remote CICS system.

Header file: `ICCRIDEH`

IccSysId constructors

Constructor (1)

IccSysId(const char* *name*)

name
The 4-character name of the CICS system.

Constructor (2)

The copy constructor.

IccSysId(const IccSysId& *id*)

id
A reference to an **IccSysId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccSysId& operator=(const IccSysId& *id*)

id
A reference to an existing **IccSysId** object.

operator= (2)

Sets the name of the CICS system held in the object.

IccSysId& operator=(const char* *name*)

name
The 4-character name of the CICS system.

Inherited public methods

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccSystem class

This is a singleton class that represents the CICS system. It is used by an application program to discover information about the CICS system on which it is running.

IccBase
  IccResource
    IccSystem

Header file: ICCSYSEH

Sample: ICC$SYS

IccSystem constructor (protected)

Constructor

IccSystem()

Public methods

These are the public methods in this class.

applName

Returns the 8-character name of the CICS region.

const char* applName()

Conditions

INVREQ

beginBrowse (1)

void beginBrowse (ResourceType *resource*,
const IccResourceId* *resId* = 0)

resource
An enumeration, defined in this class, that indicates the type of resource to be browsed within the CICS system.

resId
An optional pointer to an **IccResourceId** object that indicates the starting point for browsing through the resources.

beginBrowse (2)

Signals the start of a browse through a set of CICS resources.

void beginBrowse (ResourceType *resource*,
const char* *resName*)

resource
An enumeration, defined in this class, that indicates the type of resource to be browsed within the CICS system.

resName
The name of the resource that is to be the starting point for browsing the resources.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

dateFormat

Returns the default dateFormat for the CICS region.

const char* dateFormat()

Conditions

INVREQ

endBrowse

Signals the end of a browse through a set of CICS resources.

void endBrowse(ResourceType *resource*)

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

freeStorage

Releases the storage obtained by the **IccSystem getStorage** method.

void freeStorage(void* *pStorage*)

Conditions

INVREQ

getFile (1)

IccFile* getFile(const IccFileId& *id*)

id
A reference to an **IccFileId** object that identifies a CICS file.

getFile (2)

Returns a pointer to the **IccFile** object identified by the argument.

IccFile* getFile(const char* *fileName*)

fileName
The name of a CICS file.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

getNextFile

This method is only valid after a successful **beginBrowse(IccSystem::file)** call. It returns the next file object in the browse sequence in the CICS system.

IccFile* getNextFile()

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

getStorage

Obtains a block of storage of the requested size and returns a pointer to it. The storage is not released automatically at the end of task; it is only released when a **freeStorage** operation is performed.

void* getStorage (unsigned long *size*,
char *initByte* = -1,
unsigned long *storageOpts* = 0)

size
The amount of storage being requested, in bytes

initByte
The initial setting of all bytes in the allocated storage

storageOpts
An enumeration, defined in **IccTask** class, that affects the way that CICS allocates storage.

Conditions

LENGERR, NOSTG

instance

Returns a pointer to the singleton **IccSystem** object. The object is created if it does not already exist.

static IccSystem* instance()

operatingSystem

char operatingSystem()

Returns a 1-character value that identifies the operating system under which CICS is running:

A
AIX

N
Windows

X
z/OS

Conditions

NOTAUTH

operatingSystemLevel

Returns a halfword binary field giving the release number of the operating system under which CICS is running. The value returned is ten times the formal release number (the version number is not represented). For example, MVS/ESA Version 3 Release 2.1 would produce a value of 21.

unsigned short operatingSystemLevel()

Conditions

NOTAUTH

IccSystem public method: release

Returns the level of the CICS system. The value is taken from the number returned in the **RELEASE** parameter of the **EXE CICS INQUIRE SYSTEM** command.

For example, the release level returned for CICS Transaction Server for z/OS Version 4 Release 2 is 670.

unsigned long release()

Conditions

NOTAUTH

releaseText

Returns the same as **release**, except as a 4-character string. For example, CICS Transaction Server for z/OS [Version 1] Release 3 would return "0130".

const char* releaseText()

Conditions

NOTAUTH

sysId

Returns a reference to the **IccSysId** object that identifies this CICS system.

IccSysId& sysId()

Conditions

INVREQ

workArea

Returns a reference to the **IccBuf** object that holds the work area for the CICS system.

const IccBuf& workArea()

Conditions

INVREQ

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource

Method	Class
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

ResourceType

- autoInstallModel
- connection
- dataQueue
- exitProgram
- externalDataSet
- file
- journal
- modename
- partner
- profile
- program
- requestId
- systemDumpCode
- tempStore
- terminal
- transactionDumpCode
- transaction
- transactionClass

IccTask class

IccTask is a singleton class used to invoke task related CICS services.

IccBase
  IccResource
    IccTask

Header file: ICCTSKEH

Sample: ICC$TSK

IccTask Constructor (protected)

Constructor

IccTask()

Public methods

These are the public methods in this class.

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method in"abendCode" on page 60.

abend

Requests CICS to abend this task.

void abend (const char* *abendCode* = 0,
  AbendHandlerOpt *opt1* = respectAbendHandler**,**
  AbendDumpOpt *opt2* = createDump**)**

abendCode
The 4-character abend code

opt1
An enumeration, defined in this class, that indicates whether to respect or ignore any abend handling program specified by **setAbendHandler** method in **IccControl** class

opt2
An enumeration, defined in this class, that indicates whether a dump is to be created.

abendData

IccAbendData* abendData()

Returns a pointer to an **IccAbendData** object that contains information about the program abends, if any, that relate to this task.

commitUOW

void commitUOW()

Commit the resource updates within the current UOW for this task. This also causes a new UOW to start for subsequent resource update activity.

Conditions

INVREQ, ROLLEDBACK

delay

Requests that this task be delayed for an interval of time, or until a specific time.

void delay (const IccTime& *time*,
const IccRequestId* *reqId* = 0)

time
A reference to an object that contains information about the delay time. The object can be one of these types:

IccAbsTime
Expresses time as the number of milliseconds since the beginning of the year 1900.

IccTimeInterval
Expresses an interval of time, such as 3 hours, 2 minutes, and 1 second.

IccTimeOfDay
Expresses a time of day, such as 13 hours, 30 minutes (1-30 pm).

reqId
An optional pointer to an **IccRequestId** object that can be used to cancel an unexpired delay request.

Conditions

EXPIRED, INVREQ

dump

Requests CICS to take a memory dump for this task. (See also **setDumpOpts**.) Returns the character identifier of the dump.

const char* dump (const char* *dumpCode*,
const IccBuf* *buf* = 0)

dumpCode
A 4-character label that identifies this dump

buf
A pointer to the **IccBuf** object that contains additional data to be included in the dump.

Conditions

INVREQ, IOERR, NOSPACE, NOSTG, NOTOPEN, OPENERR, SUPPRESSED

enterTrace

Writes a user trace entry in the CICS trace table.

void enterTrace (unsigned short *traceNum*,
const char* *resource* = 0,
IccBuf* *data* = 0,
TraceOpt *opt* = normal**)**

traceNum
The trace identifier for a user trace table entry; a value in the range 0 through 199.

resource
An 8-character name to be entered in the resource field of the trace table entry.

data
A pointer to the **IccBuf** object containing data to be included in the trace record.

opt
An enumeration, defined in this class, that indicates whether tracing should be normal or whether only exceptions should be traced.

Conditions

INVREQ, LENGERR

facilityType

Returns an enumeration, defined in this class, that indicates what type of principal facility this task has. This is usually a terminal, such as when the task was started by someone keying a transaction name on a CICS terminal. It is a session if the task is the back end of a mapped APPC conversation.

FacilityType facilityType()

Conditions

INVREQ

freeStorage

Releases the storage obtained by the **IccTask getStorage** method.

void freeStorage(void* pStorage)

Conditions

INVREQ

getStorage

Obtains a block of storage of the requested size. The storage is released automatically at the end of task, or when the **freeStorage** operation is performed. See also **getStorage** on page "getStorage" on page 196 in **IccSystem**class.

void* getStorage (unsigned long *size*,
char *initByte* = -1,
unsigned short *storageOpts* = 0)

size
The amount of storage being requested, in bytes

initByte
The initial setting of all bytes in the allocated storage

storageOpts
An enumeration, defined in this class, that affects the way that CICS allocates storage.

Conditions

LENGERR, NOSTG

instance

Returns a pointer to the singleton **IccTask** object. The object is created if it does not already exist.

static IccTask* instance();

isCommandSecurityOn

Icc::Bool isCommandSecurityOn()

Returns a boolean, defined in **Icc** structure, that indicates whether this task is subject to command security checking.

Conditions

INVREQ

isCommitSupported

Returns a boolean, defined in **Icc** structure that indicates whether this task can support the **commit** method. This method returns true in most environments; the exception to this is in a DPL environment (see **link** on page “link” on page 151 in **IccProgram**).

Icc::Bool isCommitSupported()

Conditions

INVREQ

isResourceSecurityOn

Returns a boolean, defined in **Icc** structure, that indicates whether this task is subject to resource security checking.

Icc::Bool isResourceSecurityOn()

Conditions

INVREQ

isRestarted

Returns a boolean, defined in **Icc** structure, that indicates whether this task has been automatically restarted by CICS.

Icc::Bool isRestarted()

Conditions

INVREQ

isStartDataAvailable

Returns a boolean, defined in **Icc** structure, that indicates whether start data is available for this task. See the **retrieveData** method in **IccStartRequestQ** class if start data is available.

Icc::Bool isStartDataAvailable()

Conditions

INVREQ

number

Returns the number of this task, unique within the CICS system.

unsigned long number() const

principalSysId

IccSysId& principalSysId(Icc::GetOpt *opt* = Icc::object**)**

Returns a reference to an **IccSysId** object that identifies the principal system identifier for this task.

Conditions

INVREQ

priority

Returns the priority for this task.

unsigned short priority(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

rollBackUOW

Roll back (backout) the resource updates associated with the current UOW within this task.

void rollBackUOW()

Conditions

INVREQ, ROLLEDBACK

setDumpOpts

Set the dump options for this task. This method affects the behavior of the **dump** method defined in this class.

void setDumpOpts(unsigned long *opts* = dDefault)

opts
An integer, made by adding or logically ORing values from the **DumpOpts** enumeration, defined in this class.

setPriority

Changes the dispatch priority of this task.

void setPriority(unsigned short *pri*)

pri
The new priority.

Conditions

INVREQ

setWaitText

Sets the text that will appear when someone inquires on this task while it is suspended as a result of a **waitExternal** or **waitOnAlarm** method call.

void setWaitText(const char* *name*)

name
The 8-character string label that indicates why this task is waiting.

startType

StartType startType()

Returns an enumeration, defined in this class, that indicates how this task was started.

Conditions

INVREQ

suspend

Suspend this task, allowing other tasks to be dispatched.

void suspend()

transId

const IccTransId& transId()

Returns the **IccTransId** object representing the transaction name of this CICS task.

triggerDataQueueId

const IccDataQueueId& triggerDataQueueId()

Returns a reference to the **IccDataQueueId** representing the trigger queue, if this task was started as a result of data arriving on an **IccDataQueue**. See **startType** method.

Conditions

INVREQ

userId

Returns the ID of the user associated with this task.

const IccUserId& userId(Icc::GetOpt *opt* = Icc::object)

opt
An enumeration, defined in **Icc** structure, that indicates whether the information already existing in the object is to be used or whether it is to be refreshed from CICS.

Conditions

INVREQ

waitExternal

Waits for events that post Event Control Blocks (ECBs).

The call causes the issuing task to be suspended until one of the ECBs has been posted—that is, one of the events has occurred. The task can wait on more than one ECB and can be dispatched as soon as any of them are posted. For more information about ECB, see WAIT EXTERNAL.

void waitExternal (long *ECBList*,**
unsigned long *numEvents*,
WaitPurgeability *opt* = purgeable**,**
WaitPostType *type* = MVSPost**)**

ECBList
A pointer to a list of addresses of ECBs that represent events.

numEvents
The number of events in *ECBList*.

opt
An enumeration, defined in this class, that indicates whether the wait is purgeable.

type
An enumeration, defined in this class, that indicates whether the post type is a standard MVS POST.

Conditions

INVREQ

waitOnAlarm

Suspends the task until the alarm goes off (expires).

See also "setAlarm" on page 89 in **IccClock**.

void waitOnAlarm(const IccAlarmRequestId& *id*)

id
A reference to the **IccAlarmRequestId** object that identifies a particular alarm request.

Conditions

INVREQ

workArea

Returns a reference to the **IccBuf** object that holds the work area for this task.

IccBuf& workArea()

Conditions

INVREQ

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

AbendHandlerOpt

respectAbendHandler
Allows control to be passed to an abend handling program if one is in effect.

ignoreAbendHandler
Does not allow control to be passed to any abend handling program that may be in effect.

AbendDumpOpt

createDump
Take a transaction dump when servicing an abend request.

suppressDump
Do not take a transaction dump when servicing an abend request.

DumpOpts

The values may be added, or bitwise ORed, together to get the intended combination.

The values may be added, or bitwise ORed, together to get the intended combination. For example `IccTask::dProgram + IccTask::dDCT + IccTask::dSIT`.

dDefault

dComplete

dTask

dStorage

dProgram

dTerminal

dTables

dDCT

dFCT

dPCT

dPPT

dSIT

dTCT

dTRT

FacilityType

none
The task has no principal facility, that is, it is a background task.

terminal
This task has a terminal as its principal facility.

session
This task has a session as its principal facility, that is, it was probably started as a back-end DTP program.

dataqueue
This task has a transient data queue as its principal facility.

StartType

DPL
Distributed program link request

dataQueueTrigger
Trigger by data arriving on a data queue

startRequest
Started as a result of an asynchronous start request. See **IccStartRequestQ** class.

FEPIRequest
Front end programming interface.

terminalInput
Started via a terminal input

CICSInternalTask
Started by CICS.

StorageOpts

ifSOSReturnCondition
If insufficient space is available, return NOSTG condition instead of blocking the task.

below
Allocate storage below the 16Mb line.

userDataKey
Allocate storage in the USER data key.

CICSDataKey
Allocate storage in the CICS data key.

TraceOpt

normal
The trace entry is a standard entry.

exception
The trace entry is an exception entry.

WaitPostType

MVSPost
ECB is posted using the MVS POST service.

handPost
ECB is hand posted (that is, using some method other than the MVS POST service).

WaitPurgeability

purgeable
Task can be purged via a system call.

notPurgeable
Task cannot be purged via a system call.

IccTempStore class

IccTempStore objects are used to manage the temporary storage of data.

IccBase
IccResource
IccTempStore

(**IccTempStore** data can exist between transaction calls.)

Header file: `ICCTMPEH`

Sample: `ICC$TMP`

IccTempStore constructors

Constructor (1)

IccTempStore (const IccTempStoreId& *id*,
Location *loc* = auxStorage)

id
Reference to an **IccTempStoreId** object

loc
An enumeration, defined in this class, that indicates where the storage is to be located when it is first created. The default is to use auxiliary storage (disk).

Constructor (2)

IccTempStore (const char* *storeName*,
Location *loc* = auxStorage)

storeName
Specifies the 8-character name of the queue to be used. The name must be unique within the CICS system.

loc
An enumeration, defined in this class, that indicates where the storage is to be located when it is first created. The default is to use auxiliary storage (disk).

Public methods

These are the public methods in this class.

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method in“abendCode” on page 60.

clear

A synonym for **empty**. See Polymorphic Behavior for information on polymorphism.

virtual void clear()

empty

void empty()

Deletes all the temporary data associated with the **IccTempStore** object and deletes the associated TD queue.

Conditions

INVREQ, ISCINVREQ, NOTAUTH, QIDERR, SYSIDERR

get

A synonym for **readNextItem**. See Polymorphic Behavior for information on polymorphism.

virtual const IccBuf& get()

numberOfItems

unsigned short numberOfItems() const

Returns the number of items in temporary storage. This is only valid after a successful **writeItem** call.

put

A synonym for **writeItem**. See Polymorphic Behavior for information on polymorphism.

virtual void put(const IccBuf& *buffer*)

buffer
A reference to an **IccBuf** object that contains the data that is to be added to the end of the temporary storage queue.

readItem

Reads the specified item from the temporary storage queue and returns a reference to the **IccBuf** object that contains the information.

const IccBuf& readItem(unsigned short *itemNum*)

itemNum
Specifies the item number of the logical record to be retrieved from the queue.

Conditions

INVREQ, IOERR, ISCINVREQ, ITEMERR, LENGERR, NOTAUTH, QIDERR, SYSIDERR

readNextItem

Reads the next item from a temporary storage queue and returns a reference to the **IccBuf** object that contains the information.

const IccBuf& readNextItem()

Conditions

INVREQ, IOERR, ISCINVREQ, ITEMERR, LENGERR, NOTAUTH, QIDERR, SYSIDERR

rewriteItem

The parameters are:This method updates the specified item in the temporary storage queue.

void rewriteItem (unsigned short *itemNum*,
const IccBuf& *item*,
NoSpaceOpt *opt* = suspend**)**

itemNum
Specifies the item number of the logical record that is to be modified

item
The name of the **IccBuf** object that contains the update data.

opt
An enumeration, defined in this class, that indicates whether the application program is to be suspended if a shortage of space in the queue prevents the record being added. suspend is the default.

Conditions

INVREQ, IOERR, ISCINVREQ, ITEMERR, LENGERR, NOSPACE, NOTAUTH, QIDERR, SYSIDERR

writeItem (1)

unsigned short writeItem (const IccBuf& *item*,
NoSpaceOpt *opt* = suspend**)**

item
The name of the **IccBuf** object that contains the data that is to added to the end of the temporary storage queue.

opt
An enumeration, defined in this class, that indicates whether the application program is to be suspended if a shortage of space in the queue prevents the record being added. suspend is the default.

writeItem (2)

This method adds a new record at the end of the temporary storage queue. The returned value is the item number that was created (if this was done successfully).

unsigned short writeItem (const char* *text*,
NoSpaceOpt *opt* = suspend**)**

text
The text string that is to added to the end of the temporary storage queue.

opt
An enumeration, defined in this class, that indicates whether the application program is to be suspended if a shortage of space in the queue prevents the record being added. suspend is the default.

Conditions

INVREQ, IOERR, ISCINVREQ, ITEMERR, LENGERR, NOSPACE, NOTAUTH, QIDERR, SYSIDERR

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase

Method	Class
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

Location

auxStorage
Temporary store data is to reside in auxiliary storage (disk).

memory
Temporary store data is to reside in memory.

NoSpaceOpt

Take this action if a shortage of space in the queue prevents the record being added immediately.

suspend
Suspend the application program.

returnCondition
Do not suspend the application program, but raise the NOSPACE condition instead.

IccTempStoreId class

IccTempStoreId class is used to identify a temporary storage name in the CICS system.

IccBase
  IccResourceId
    IccTempStoreId

Header file: ICCRIDEH

IccTempStoreId constructors

Constructor (1)

IccTempStoreId(const char* *name*)

name
The 8-character name of the temporary storage entry.

Constructor (2)

The copy constructor.

IccTempStoreId(const IccTempStoreId& *id*)

id
A reference to an **IccTempStoreId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccTempStoreId& operator=(const char* *name*)

name
The 8-character name of the temporary storage entry.

operator= (2)

Assigns a new value.

IccTempStoreId& operator=(const IccTempStoreId& *id*)

id
A reference to an **IccTempStoreId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccTermId class

IccTermId class is used to identify a terminal name in the CICS system.

IccBase
  IccResourceId
    IccTermId

Header file: `ICCRIDEH`

IccTermId constructors

Constructor (1)

IccTermId(const char* *name*)

name
The 4-character name of the terminal

Constructor (2)

The copy constructor.

IccTermId(const IccTermId& *id*)

id
A reference to an **IccTermId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccTermId& operator=(const char* *name*)

name
The 4-character name of the terminal

operator= (2)

Assigns a new value.

IccTermId& operator=(const IccTermId& *id*)

id
A reference to an **IccTermId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccTerminal class

This is a singleton class that represents the terminal that belongs to the CICS task. It can only be created if the transaction has a 3270 terminal as its principal facility, otherwise an exception is thrown.

IccBase
  IccResource
    IccTerminal

Header file: `ICCTRMEH`

Sample: `ICC$TRM`

IccTerminal constructor (protected)

Constructor

IccTerminal()

Public methods

These are the public methods in this class.

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method in“abendCode” on page 60.

AID

Returns an enumeration, defined in this class, that indicates which AID (action identifier) key was last pressed at this terminal.

AIDVal AID()

clear

virtual void clear()

A synonym for **erase**. See Polymorphic Behavior for information on polymorphism.

cursor

unsigned short cursor()

Returns the current cursor position as an offset from the upper-left corner of the screen.

data

IccTerminalData* data()

Returns a pointer to an **IccTerminalData** object that contains information about the characteristics of the terminal. The object is created if it does not already exist.

erase

void erase()

Erase all the data displayed at the terminal.

Conditions

INVREQ, INVPARTN

freeKeyboard

Frees the keyboard so that the terminal can accept input.

void freeKeyboard()

Conditions

INVREQ, INVPARTN

get

A synonym for **receive**. See Polymorphic Behavior for information on polymorphism.

virtual const IccBuf& get()

height

unsigned short height(Icc::getopt *opt* = Icc::object)

Returns how many lines the screen holds.

Conditions

INVREQ

inputCursor

Returns the position of the cursor on the screen.

unsigned short inputCursor()

instance

static IccTerminal* instance()

Returns a pointer to the single **IccTerminal** object. The object is created if it does not already exist.

line

unsigned short line()

Returns the current line number of the cursor from the beginning of the screen.

netName

const char* netName()

Returns the 8-byte string representing the network logical unit name of the principal facility.

operator« (1)

Sets the foreground color for data subsequently sent to the terminal.

IccTerminal& operator « (Color *color*)

operator« (2)

Sets the highlighting used for data subsequently sent to the terminal.

IccTerminal& operator « (Highlight *highlight*)

operator« (3)

Writes another buffer.

IccTerminal& operator « (const IccBuf& *buffer*)

operator« (4)

Writes a character.

IccTerminal& operator « (char *ch*)

operator« (5)

Writes a character.

IccTerminal& operator « (signed char *ch*)

operator« (6)

Writes a character.

IccTerminal& operator « (unsigned char *ch*)

operator« (7)

Writes a string.

IccTerminal& operator « (const char* *text*)

operator« (8)

Writes a string.

IccTerminal& operator « (const signed char* *text*)

operator« (9)

Writes a string.

IccTerminal& operator « (const unsigned char* *text*)

operator« (10)

Writes a short.

IccTerminal& operator « (short *num*)

operator« (11)

Writes an unsigned short.

IccTerminal& operator « (unsigned short *num*)

operator« (12)

Writes a long.

IccTerminal& operator « (long *num*)

operator« (13)

Writes an unsigned long.

IccTerminal& operator « (unsigned long *num*)

operator« (14)

Writes an integer.

IccTerminal& operator « (int *num*)

operator« (15)

Writes a float.

IccTerminal& operator « (float *num*)

operator« (16)

Writes a double.

IccTerminal& operator « (double *num*)

operator« (17)

Writes a long double.

IccTerminal& operator « (long double *num*)

operator« (18)

IccTerminal& operator « (IccTerminal& (*f)(IccTerminal&))

Enables the following syntax:

```
Term « "Hello World" « endl;
Term « "Hello again" « flush;
```

put

virtual void put(const IccBuf& *buf*)

A synonym for **sendLine**. See Polymorphic Behavior for information on polymorphism.

receive

Receives data from the terminal

const IccBuf& receive(Case *caseOpt* = upper**)**

caseOpt
An enumeration, defined in this class, that indicates whether text is to be converted to uppercase.

Conditions

EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

receive3270Data

Receives the 3270 data buffer from the terminal

const IccBuf& receive3270Data(Case *caseOpt* = upper**)**

caseOpt
An enumeration, defined in this class, that indicates whether text is to be converted to uppercase.

Conditions

INVREQ, LENGERR, TERMERR

send (1)

void send(const IccBuf& *buffer*)

buffer
A reference to an **IccBuf** object that holds the data that is to be sent.

send (2)

void send (const char* *format*,
...)

format
A format string, as in the **printf** standard library function.

...
The optional arguments that accompany *format*.

send (3)

void send (unsigned short *row*,
unsigned short *col*,
const IccBuf& *buffer*)

row
The row where the writing of the data is started.

col
The column where the writing of the data is started.

buffer
A reference to an **IccBuf** object that holds the data that is to be sent.

send (4)

Writes the specified data to either the current cursor position or to the cursor position specified by the arguments.

void send (unsigned short *row*,
unsigned short *col*,
const char* *format*,
...)

row
The row where the writing of the data is started.

col
The column where the writing of the data is started.

format
A format string, as in the **printf** standard library function.

...
The optional arguments that accompany *format*.

Conditions

INVREQ, LENGERR, TERMERR

send3270Data (1)

void send3270Data(const IccBuf& *buffer*)

buffer
A reference to an **IccBuf** object that holds the data that is to be sent.

send3270Data (2)

void send3270 Data(const char* *format*,
...)

format
A format string, as in the **printf** standard library function

...
The optional arguments that accompany *format*.

send3270Data (3)

void send3270Data (unsigned short *col*,
const IccBuf& *buf*)

col
The column where the writing of the data is started

buffer
A reference to an **IccBuf** object that holds the data that is to be sent.

send3270Data (4)

Writes the specified data to either the next line of the terminal or to the specified column of the current line.

void send3270Data (unsigned short *col*,
const char* *format*,
...)

col
The column where the writing of the data is started

format
A format string, as in the **printf** standard library function

...
The optional arguments that accompany *format*.

Conditions

INVREQ, LENGERR, TERMERR

sendLine (1)

void sendLine(const IccBuf&*buffer*)

buffer
A reference to an **IccBuf** object that holds the data that is to be sent.

sendLine (2)

void sendLine (const char* *format*,
...)

format
A format string, as in the **printf** standard library function

...
The optional arguments that accompany *format*.

sendLine (3)

void sendLine (unsigned short *col*,
const IccBuf& *buf*)

col
The column where the writing of the data is started

buffer
A reference to an **IccBuf** object that holds the data that is to be sent.

sendLine (4)

Writes the specified data to either the next line of the terminal or to the specified column of the current line.

void sendLine (unsigned short *col*,
const char* *format*,
...)

col
The column where the writing of the data is started

format
A format string, as in the **printf** standard library function

...
The optional arguments that accompany *format*.

Conditions

INVREQ, LENGERR, TERMERR

setColor

Changes the color of the text subsequently sent to the terminal.

void setColor(Color *color*=defaultColor)

color
An enumeration, defined in this class, that indicates the color of the text that is written to the screen.

setCursor (1)

void setCursor(unsigned short *offset*)

offset
The position of the cursor where the upper-left corner is 0.

setCursor (2)

Two different ways of setting the position of the cursor on the screen.

void setCursor (unsigned short *row*,
unsigned short *col*)

row
The row number of the cursor where the top row is 1

col
The column number of the cursor where the left column is 1

Conditions

INVREQ, INVPARTN

setHighlight

Changes the higlighting of the data subsequently sent to the terminal.

void setHighlight(Highlight *highlight* = normal)

highlight
An enumeration, defined in this class, that indicates the highlighting of the text that is written to the screen.

setLine

Moves the cursor to the start of line *lineNum*, where 1 is the first line of the terminal. The default is to move the cursor to the start of line 1.

void setLine(unsigned short *lineNum* = 1)

lineNum
The line number, counting from the start.

Conditions

INVREQ, INVPARTN

setNewLine

Requests that *numLines* blank lines be sent to the terminal.

void setNewLine(unsigned short *numLines* = 1)

numLines
The number of blank lines.

Conditions

INVREQ, INVPARTN

setNextCommArea

Specifies the COMMAREA that is to be passed to the next transaction started on this terminal.

void setNextCommArea(const IccBuf& *commArea*)

commArea
A reference to the buffer that is to be used as a COMMAREA.

setNextInputMessage

Specifies data that is to be made available, by the **receive** method, to the next transaction started at this terminal.

void setNextInputMessage(const IccBuf& *message*)

message
A reference to the buffer that holds the input message.

setNextTransId

Specifies the next transaction that is to be started on this terminal.

void setNextTransId (const IccTransId& *transid*,
NextTransIdOpt *opt* = queue**)**

transid
A reference to the **IccTransId** object that holds the name of a transaction

opt
An enumeration, defined in this class, that indicates whether *transId* should be queued or started immediately (that is, it should be the very next transaction) at this terminal.

signoff

void signoff()

Signs off the user who is currently signed on. Authority reverts to the default user.

Conditions

INVREQ

signon (1)

void signon (const IccUserId& *id*,
const char* *password* = 0,
const char* *newPassword* = 0)

id
A reference to an **IccUserId** object

password
The 8-character existing password.

newPassword
An optional 8-character new password.

signon (2)

Signs the user on to the terminal.

void signon (IccUser& *user*,
const char* *password* = 0,
const char* *newPassword* = 0)

user
A reference to an **IccUser** object

password
The 8-character existing password.

newPassword
An optional 8-character new password. This method differs from the first **signon** method in that the **IccUser** object is interrogated to discover **IccGroupId** and language information. The object is also updated with language and ESM return and response codes.

Conditions

INVREQ, NOTAUTH, USERIDERR

waitForAID (1)

Waits for any input and returns an enumeration, defined in this class, that indicates which AID key is expected.

AIDVal waitForAID()

waitForAID (2)

Waits for the specified AID key to be pressed, before returning control. This method loops, receiving input from the terminal, until the correct AID key is pressed by the operator.

void waitForAID(AIDVal *aid*)

aid
An enumeration, defined in this class, that indicates which AID key was last pressed.

Conditions

EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

width

Returns the width of the screen in characters.

unsigned short width(Icc::getopt *opt* = Icc::object**)**

Conditions

INVREQ

workArea

Returns a reference to the **IccBuf** object that holds the terminal work area.

IccBuf& workArea()

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

AIDVal

ENTER

CLEAR

PA1 to PA3

PF1 to PF24

Case

upper

mixed

Color

defaultColor

blue

red

pink

green

cyan

yellow

neutral

Highlight

defaultHighlight

blink

reverse

underscore

NextTransIdOpt

queue
Queue the transaction with any other outstanding starts queued on the terminal.

immediate
Start the transaction immediately, that is, before any other outstanding starts queued on the terminal.

IccTerminalData class

IccTerminalData is a singleton class owned by **IccTerminal**. It contains information about the terminal characteristics.

See "data" on page 220 in **IccTerminal** class).

IccBase
  IccResource
    IccTerminalData

Header file: `ICCTMDEH`

Sample: `ICC$TRM`

IccTerminalData constructor (protected)

Constructor

IccTerminalData()

Public methods

These are the public methods in this class.

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method in"abendCode" on page 60.

alternateHeight

Returns the alternate height of the screen, in lines.

unsigned short alternateHeight(Icc::GetOpt *opt* = Icc::object**)**

opt
An enumeration that indicates whether the information in the object should be refreshed from CICS before being extracted. The default is not to refresh.

Conditions

INVREQ

alternateWidth

Returns the alternate width of the screen, in characters.

unsigned short alternateWidth(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

defaultHeight

Returns the default height of the screen, in lines.

unsigned short defaultHeight(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

defaultWidth

Returns the default width of the screen, in characters.

unsigned short defaultWidth(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

graphicCharCodeSet

Returns the binary code page global identifier as a value in the range 1 to 65534, or 0 for a non-graphics terminal.

unsigned short graphicCharCodeSet(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

graphicCharSetId

Returns the graphic character set global identifier as a number in the range 1 to 65534, or 0 for a non-graphics terminal.

unsigned short graphicCharSetId(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isAPLKeyboard

Returns a boolean that indicates whether the terminal has the APL keyboard feature.

Icc::Bool isAPLKeyboard(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isAPLText

Returns a boolean that indicates whether the terminal has the APL text feature.

Icc::Bool isAPLText(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isBTrans

Returns a boolean that indicates whether the terminal has the background transparency capability.

Icc::Bool isBTrans(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isColor

Returns a boolean that indicates whether the terminal has the extended color capability.

Icc::Bool isColor(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isEWA

Returns a Boolean that indicates whether the terminal supports Erase Write Alternative.

Icc::Bool isEWA(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isExtended3270

Returns a Boolean that indicates whether the terminal supports the 3270 extended data stream.

Icc::Bool isExtended3270(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isFieldOutline

Returns a boolean that indicates whether the terminal supports field outlining.

Icc::Bool isFieldOutline(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isGoodMorning

Returns a boolean that indicates whether the terminal has a 'good morning' message.

Icc::Bool isGoodMorning(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isHighlight

Returns a boolean that indicates whether the terminal has extended highlight capability.

Icc::Bool isHighlight(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isKatakana

Returns a boolean that indicates whether the terminal supports Katakana.

Icc::Bool isKatakana(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isMSRControl

Returns a boolean that indicates whether the terminal supports magnetic slot reader control.

Icc::Bool isMSRControl(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isPS

Returns a boolean that indicates whether the terminal supports programmed symbols.

Icc::Bool isPS(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isSOSI

Returns a boolean that indicates whether the terminal supports mixed EBCDIC/DBCS fields.

Icc::Bool isSOSI(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isTextKeyboard

Returns a boolean that indicates whether the terminal supports TEXTKYBD.

Icc::Bool isTextKeyboard(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isTextPrint

Returns a boolean that indicates whether the terminal supports TEXTPRINT.

Icc::Bool isTextPrint(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

isValidation

Returns a boolean that indicates whether the terminal supports validation.

Icc::Bool isValidation(Icc::GetOpt *opt* = Icc::object**)**

Conditions

INVREQ

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccTime class

IccTime is used to contain time information and is the base class from which **IccAbsTime**, **IccTimeInterval**, and **IccTimeOfDay** classes are derived.

IccBase
  IccResource
    IccTime

Header file: ICCTIMEH

IccTime constructor (protected)

Constructor

IccTime (unsigned long *hours* = 0,
unsigned long *minutes* = 0,
unsigned long *seconds* = 0)

hours
The number of hours

minutes
The number of minutes

seconds
The number of seconds

Public methods

These are the public methods in this class.

hours

Returns the hours component of time—the value specified in the constructor.

virtual unsigned long hours() const

minutes

virtual unsigned long minutes() const

Returns the minutes component of time—the value specified in the constructor.

seconds

virtual unsigned long seconds() const

Returns the seconds component of time—the value specified in the constructor.

timeInHours

virtual unsigned long timeInHours()

Returns the time in hours.

timeInMinutes

virtual unsigned long timeInMinutes()

Returns the time in minutes.

timeInSeconds

virtual unsigned long timeInSeconds()

Returns the time in seconds.

type

Type type() const

Returns an enumeration, defined in this class, that indicates what type of subclass of **IccTime** this is.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
isEDFOn	IccResource
operator delete	IccBase

Method	Class
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

Type

absTime
The object is of **IccAbsTime** class. It is used to represent a current date and time as the number of milliseconds that have elapsed since the beginning of the year 1900.

timeInterval
The object is of **IccTimeInterval** class. It is used to represent a length of time, such as 5 minutes.

timeOfDay
The object is of **IccTimeOfDay** class. It is used to represent a particular time of day, such as midnight.

IccTimeInterval class

This class holds information about a time interval.

IccBase
IccResource
IccTime
IccTimeInterval

Header file: `ICCTIMEH`

IccTimeInterval constructors

Constructor (1)

IccTimeInterval (unsigned long *hours* = 0,
unsigned long *minutes* = 0,
unsigned long *seconds* = 0)

hours
The initial hours setting. The default is 0.

minutes
The initial minutes setting. The default is 0.

seconds
The initial seconds setting. The default is 0.

Constructor (2)

The copy constructor.

IccTimeInterval(const IccTimeInterval& *time*)

Public methods

These are the public methods in this class.

operator=

Assigns one **IccTimeInterval** object to another.

IccTimeInterval& operator=(const IccTimeInterval& *timeInterval*)

set

Changes the time held in the **IccTimeInterval** object.

void set (unsigned long *hours*,
unsigned long *minutes*,
unsigned long *seconds*)

hours
The new hours setting

minutes
The new minutes setting

seconds
The new seconds setting

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource

Method	Class
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
hours	IccTime
isEDFOn	IccResource
minutes	IccTime
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
timeInHours	IccTime
timeInMinutes	IccTime
timeInSeconds	IccTime
type	IccTime

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccTimeOfDay class

This class holds information about the time of day.

IccBase
  IccResource
    IccTime
      IccTimeOfDay

Header file: `ICCTIMEH`

IccTimeOfDay constructors

Constructor (1)

IccTimeOfDay (unsigned long *hours* = 0,
unsigned long *minutes* = 0,
unsigned long *seconds* = 0)

hours
The initial hours setting. The default is 0.

minutes
The initial minutes setting. The default is 0.

seconds
The initial seconds setting. The default is 0.

Constructor (2)

The copy constructor

IccTimeOfDay(const IccTimeOfDay& *time*)

Public methods

These are the public methods in this class.

operator=

Assigns one **IccTimeOfDay** object to another.

IccTimeOfDay& operator=(const IccTimeOfDay& *timeOfDay*)

set

Changes the time held in the **IccTimeOfDay** object.

void set (unsigned long *hours*,
unsigned long *minutes*,
unsigned long *seconds*)

hours
The new hours setting

minutes
The new minutes setting

seconds
The new seconds setting

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
hours	IccTime
isEDFOn	IccResource
minutes	IccTime
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
timeInHours	IccTime
timeInMinutes	IccTime
timeInSeconds	IccTime
type	IccTime

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccTPNameId class

IccTPNameId class holds a 1-64 byte TP partner name.

IccBase
IccResourceId
IccTPNameId

IccTPNameId class holds a 1-64 byte TP partner name.

Header file: ICCRIDEH

IccTPNameId constructors

Constructor (1)

IccTPNameId(const char* *name*)

name
The 1- to 64-character TP name.

Constructor (2)

The copy constructor.

IccTPNameId(const IccTPNameId& *id*)

id
A reference to an **IccTPNameId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccTPNameId& operator=(const char* *name*)

name
The 1- to 64-character TP name.

operator= (2)

Assigns a new value.

IccTPNameId& operator=(const IccTPNameId& *id*)

id
A reference to an **IccTPNameId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccTransId class

IccTransId class identifies a transaction name in the CICS system.

IccBase
  IccResourceId
    IccTransId

Header file: `ICCRIDEH`

IccTransId constructors

Constructor (1)

IccTransId(const char* *name*)

name
The 4-character transaction name.

Constructor (2)

The copy constructor.

IccTransId(const IccTransId& *id*)

id
A reference to an **IccTransId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccTransId& operator=(const char* *name*)

name
The 4-character transaction name.

operator= (2)

Assigns a new value.

IccTransId& operator=(const IccTransId& *id*)

id
A reference to an **IccTransId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccUser class

This class represents a CICS user.

IccBase
  IccResource
    IccUser

Header file: ICCUSREH

Sample: ICC$USR

IccUser constructors

Constructor (1)

IccUser (const IccUserId& *id*,
  const IccGroupId* *gid* = 0)

id
A reference to an **IccUserId** object that contains the user ID name

gid
An optional pointer to an **IccGroupId** object that contains information about the user's group ID.

Constructor (2)

IccUser (const char* *userName*,
  const char* *groupName* = 0)

userName
The 8-character user ID

gid
The optional 8-character group ID.

Public methods

These are the public methods in this class.

changePassword

Attempts to change the user's password.

void changePassword (const char* *password*,
const char* *newPassword*)

password
The user's existing password—a string of up to 8 characters

newPassword
The user's new password—a string of up to 8 characters.

Conditions

INVREQ, NOTAUTH, USERIDERR

daysUntilPasswordExpires

Returns the number of days before the password expires. This method is valid after a successful **verifyPassword** method call in this class.

unsigned short daysUntilPasswordExpires() const

ESMReason

unsigned long ESMReason() const

Returns the external security reason code of interest if a **changePassword** or **verifyPassword** method call is unsuccessful.

ESMResponse

unsigned long ESMResponse() const

Returns the external security response code of interest if a **changePassword** or **verifyPassword** method call is unsuccessful.

groupId

const IccGroupId& groupId() const

Returns a reference to the **IccGroupId** object that holds information on the user's group ID.

invalidPasswordAttempts

unsigned long invalidPasswordAttempts() const

Returns the number of times the wrong password has been entered for this user since the last successful signon. This method should only be used after a successful **verifyPassword** method.

language

const char* language() const

Returns the user's language after a successful call to **signon** in **IccTerminal**.

lastPasswordChange

const IccAbsTime& lastPasswordChange() const

Returns a reference to an **IccAbsTime** object that holds the time when the password was last changed. This method should only be used after a successful **verifyPassword** method.

lastUseTime

const IccAbsTime& lastUseTime() const

Returns a reference to an **IccAbsTime** object that holds the time when the user ID was last used. This method should only be used after a successful **verifyPassword** method.

passwordExpiration

const IccAbsTime& passwordExpiration() const

Returns a reference to an **IccAbsTime** object that holds the time when the password will expire. This method should only be used after a successful **verifyPassword** method.

setLanguage

void setLanguage(const char* *language*)

Sets the IBM-defined national language code that is to be associated with this user. This should be a three character value.

verifyPassword

void verifyPassword(const char* *password*)

Checks that the supplied password matches the password recorded by the external security manager for this **IccUser**.

Conditions

INVREQ, NOTAUTH, USERIDERR

Inherited public methods

These are the public methods inherited by this class.

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

IccUserId class

IccUserId class represents an 8-character user name.

IccBase
  IccResourceId
    IccUserId

IccUserId class represents an 8-character user name.

Header file: `ICCRIDEH`

IccUserId constructors

Constructor (1)

IccUserId(const char* *name*)

name
The 8-character name of the user ID.

Constructor (2)

The copy constructor.

IccUserId(const IccUserId& *id*)

id
A reference to an **IccUserId** object.

Public methods

These are the public methods in this class.

operator= (1)

IccUserId& operator=(const char* *name*)

name
The 8-character name of the user ID.

operator= (2)

Assigns a new value.

IccUserId& operator=(const IccUserId& *id*)

id
A reference to an **IccUserId** object.

Inherited public methods

These are the public methods inherited by this class.

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

Inherited protected methods

These are the protected methods inherited by this class.

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccValue structure

This structure contains CICS-value data areas (CVDAs) as an enumeration.

Header file: `ICCVALEH`

Enumeration

Listing of valid CVDAs

Valid CVDAs are listed in the CVDAs and numeric values topics in the System Programming reference information.

main function

You are recommended to include this code in your application.

It initializes the CICS Foundation Classes correctly, provides default exception handling, and releases allocated memory after it is finished. You may substitute your own variation of this **main** function, but this should rarely be necessary.

Source file: `ICCMAIN`

The stub has three functions:

1. It initializes the Foundation Classes environment. You can customize the way it does this by using `#defines` that control:
   - Memory management (see Storage management)
   - Family Subset enforcement (see "FamilySubset" on page 59)
   - EDF enablement (see Program debugging)
2. It provides a default definition of a class **IccUserControl**, derived from **IccControl**, that includes a default constructor and **run** method.
3. It invokes the **run** method of the user's control object using a try-catch construct.

The following information is the functional part of the **main** code:

```
int main()                                                  1
{
    Icc::initializeEnvironment(ICC_CLASS_MEMORY_MGMT,       2
                               ICC_FAMILY_SUBSET,
                               ICC_EDF_BOOL);
    try                                                     3
    {
        ICC_USER_CONTROL control;                           4
        control.run();                                      5
    }
    catch(IccException& exc)                                6
    {
        Icc::catchException(exc);                           7
    }
    catch(…)                                                8
    {
        Icc::unknownException();                            9
    }
    Icc::returnToCICS();                                    10
}
```

1
This is the main C++ entry point.

2
This call initializes the environment and is essential. The three parameters have previously been defined to the defaults for the platform.

3
Run the user's application code, using **try** and **catch**, in case the application code does not catch exceptions.

4
Create control object.

5
Invoke **run** method of control object (defined as pure virtual in **IccControl**.

6
Catch any **IccException** objects not caught by the application.

7
Call this function to abend task.

8
Catch any other exceptions not caught by application.

9
Call this function to abend task.

10
Return control to CICS.

Notices

This information was developed for products and services offered in the United States of America. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 rights 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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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 want 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

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119 Armonk,
NY 10504-1785
United States of America

© Copyright IBM Corp. 1974, 2023

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 Client Relationship Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

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 actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Programming interface information

IBM CICS supplies some documentation that can be considered to be Programming Interfaces, and some documentation that cannot be considered to be a Programming Interface.

Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 4 (CICS TS 5.4) are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- Securing overview
- Developing for external interfaces
- Application development reference
- Reference: system programming
- Reference: connectivity

Information that is NOT intended to be used as a Programming Interface of CICS TS 5.4, but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- Troubleshooting and support
- Reference: diagnostics

If you access the CICS documentation in manuals in PDF format, Programming Interfaces that allow the customer to write programs to obtain the services of CICS TS 5.4 are included in the following manuals:

- Application Programming Guide and Application Programming Reference
- Business Transaction Services

- Customization Guide
- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- Installation Guide
- Security Guide
- CICS Transactions
- CICSPlex® System Manager (CICSPlex SM) Managing Workloads
- CICSPlex SM Managing Resource Usage
- CICSPlex SM Application Programming Guide and Application Programming Reference
- Java™ Applications in CICS

If you access the CICS documentation in manuals in PDF format, information that is NOT intended to be used as a Programming Interface of CICS TS 5.4, but that might be misconstrued as Programming Interfaces, is included in the following manuals:

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSPlex SM Problem Determination Guide

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Apache, Apache Axis2, Apache Maven, Apache Ivy, the Apache Software Foundation (ASF) logo, and the ASF feather logo are trademarks of Apache Software Foundation.

Gradle and the Gradlephant logo are registered trademark of Gradle, Inc. and its subsidiaries in the United States and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

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

Red Hat®, and Hibernate® are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Spring Boot is a trademark of Pivotal Software, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Zowe™, the Zowe logo and the Open Mainframe Project™ are trademarks of The Linux Foundation.

The Stack Exchange name and logos are trademarks of Stack Exchange Inc.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability
These terms and conditions are in addition to any terms of use for the IBM website.

Personal use
You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use
You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights
Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM online privacy statement

IBM Software products, including software as a service solutions, (*Software Offerings*) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information (PII) is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect PII. If this Software Offering uses cookies to collect PII, specific information about this offering's use of cookies is set forth below:

For the CICSPlex SM Web User Interface (main interface):
Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name and other PII for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface (data interface):
Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other PII for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface ("hello world" page):
Depending upon the configurations deployed, this Software Offering may use session cookies that do not collect PII. These cookies cannot be disabled.

For CICS Explorer®:

Depending upon the configurations deployed, this Software Offering may use session and persistent preferences that collect each user's user name and password, for purposes of session management, authentication, and single sign-on configuration. These preferences cannot be disabled, although storing a user's password on disk in encrypted form can only be enabled by the user's explicit action to check a check box during sign-on.

If the configurations deployed for this Software Offering provide you, as customer, the ability to collect PII from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM Privacy Policy and IBM Online Privacy Statement, the section entitled *Cookies, Web Beacons and Other Technologies* and the IBM Software Products and Software-as-a-Service Privacy Statement.

Index

Special Characters

... (parameter)
    in sendLine 228

Numerics

0 (zero)
    in actionOnConditionAsChar 162

A

A
    in actionOnConditionAsChar 162
    in operatingSystem 197
abend
    in IccTask class 200
    in Parameter level 37
abend codes 32
abendCode
    in IccAbendData class 60
abendCode (parameter)
    in abend 200
abendData
    in IccTask class 200
AbendDumpOpt
    in Enumerations 209
    in IccTask class 209
AbendHandlerOpt
    in Enumerations 209
    in IccTask class 209
abendTask
    in ActionOnCondition 167
    in CICS conditions 35
absTime
    in IccClock class 87
    in Type 244
absTime (parameter)
    in Constructor 65
    in operator= 67
access
    in IccFile class 115
Access
    in Enumerations 125
    in IccFile class 125
access (parameter)
    in setAccess 122
Accessing start data
    in Starting transactions asynchronously 21
    in Using CICS Services 21
accessMethod
    in IccFile class 115
action (parameter)
    in setActionOnAnyCondition 165
    in setActionOnCondition 165
actionOnCondition
    in IccResource class 162
ActionOnCondition
    in Enumerations 167
    in IccResource class 167
actionOnConditionAsChar
    in IccResource class 162
actions (parameter)
    in setActionsOnConditions 165, 166
actionsOnConditionsText
    in IccResource class 163
addable
    in Access 125
address
    in IccProgram class 150
AID
    in IccTerminal class 219
aid (parameter)
    in waitForAID 232
AIDVal
    in Enumerations 234
    in IccTerminal class 234
AIX, CICS for
    in Platform differences 36
allocate
    in IccSession class 176
AllocateOpt
    in Enumerations 185
    in IccSession class 185
alternateHeight
    in IccTerminalData class 235
    in Public methods 235
alternateWidth
    in IccTerminalData class 236
    in Public methods 236
append
    in IccBuf class 78
applName
    in IccSystem class 194
ASRAInterrupt
    in IccAbendData class 61
    in Public methods 61
ASRAKeyType
    in IccAbendData class 61
    in Public methods 61
ASRAPSW
    in IccAbendData class 62
ASRARegisters
    in IccAbendData class 62
    in Public methods 62
ASRASpaceType
    in IccAbendData class 62
    in Public methods 62
ASRAStorageType
    in IccAbendData class 63
    in Public methods 63
assign
    in Example of file control 18
    in IccBuf class 78, 79

assign *(continued)*
in IccKey class 141
automatic
in UpdateMode 92
Automatic condition handling (callHandleEvent)
in CICS conditions 35
in Conditions, errors, and exceptions 35
automatic creation 5
automatic deletion 5
auxStorage
in Location 215

B

base class
overview 6
Base classes
in Overview of the foundation classes 6
baseName (parameter)
in NameOpt 76
BASESPACE
in ASRASpaceType 62
BDAM 15
beginBrowse
in IccSystem class 194, 195
beginInsert
in Writing records 16
beginInsert (VSAM only)
in IccFile class 115
in Public methods 115
below
in StorageOpts 210
blink
in Highlight 234
blue
in Color 234
Bool
in Enumerations 58
in Icc structure 58
BoolSet
in Enumerations 59
in Icc structure 59
boolText
in Functions 56
in Icc structure 56
browsable
in Access 125
browsing records 17
Browsing records
in File control 17
in Using CICS Services 17
buf (parameter)
in dump 201, 202
in put 224
in send3270Data 227
in sendLine 228
in setData 188, 189
buffer
in Example of starting transactions 22
buffer (parameter)
in Constructor 77
in operator!= 82
in operator« 82, 222
in operator+= 81
buffer (parameter) *(continued)*
in operator= 81
in operator== 82
in Polymorphic Behavior 39
in put 105, 134, 164, 165, 213
in registerData 187
in rewriteRecord 122
in send 225, 226
in send3270Data 226, 227
in sendLine 228
in writeRecord 124
Buffer objects
Data area extensibility 12
Data area ownership 12
IccBuf constructors 13
IccBuf methods 14
Working with IccResource subclasses 14
buffers 12, 14
byAddress
in LockType 174
byValue
in LockType 174

C

C++ exceptions 32
C++ Exceptions and the Foundation Classes
in Conditions, errors, and exceptions 32
callHandleEvent
in ActionOnCondition 167
in CICS conditions 35
calling conventions 41
Calling methods on a resource object
in Overview of the foundation classes 12
in Using CICS resources 12
callingProgramId
in IccControl class 98
in Public methods 98
cancel
in Cancelling unexpired start requests 21
in IccRequestId class 159
in IccStartRequestQ class 186
cancelAbendHandler
in IccControl class 98
cancelAlarm
in IccClock class 87
Cancelling unexpired start requests
in Starting transactions asynchronously 21
in Using CICS Services 21
Case
in Enumerations 234
in IccTerminal class 234
caseOpt (parameter)
in receive 225
in receive3270Data 225
catch
in C++ Exceptions and the Foundation Classes 32, 33
in Exception handling (throwException) 36
in main function 258
catchException
in Functions 56
in Icc structure 56
ch (parameter)
in operator« 83, 222

changePassword
in IccUser class 253
in Public methods 253
char*
in C++ Exceptions and the Foundation Classes 33
CheckOpt
in Enumerations 192
in IccStartRequestQ class 192
CICS
in ASRAStorageType 63
in GetOpt 60
in Platform differences 36
CICS conditions
abendTask 36
automatic condition handling 35
Automatic condition handling (callHandleEvent) 35
callHandleEvent 35
exception handling 36
Exception handling (throwException) 36
in Conditions, errors, and exceptions 34
manual condition handling 35
Manual condition handling (noAction) 35
noAction 35
severe error handling 36
Severe error handling (abendTask) 36
throwException 36
CICS for AIX
in Platform differences 36
CICS resources 11
CICSCondition
in C++ Exceptions and the Foundation Classes 34
in Type 113
CICSDataKey
in StorageOpts 210
CICSEXECKEY
in ASRAKeyType 61
CICSInternalTask
in StartType 210
class
base 6
resource 8
resource identification 7
singleton 11
support 10
ClassMemoryMgmt
in Enumerations 59
in Icc structure 59
className
in IccBase class 73
in IccEvent class 109
in IccException class 111
in IccMessage class 147
className (parameter)
in Constructor 111, 146
in setClassName 74
classType
in IccBase class 73
in IccEvent class 109
in IccException class 111
ClassType
in Enumerations 75
in IccBase class 75
classType (parameter)
in Constructor 111, 161
clear
in Example of polymorphic behavior 40
in IccDataQueue class 104
in IccResource class 163
in IccTempStore class 212
in IccTerminal class 219
in Polymorphic Behavior 39
CLEAR
in AIDVal 234
clearData
in IccStartRequestQ class 187
clearInputMessage
in IccProgram class 151
clearPrefix
in IccJournal class 133
closed
in Status 126
cmmCICS
in ClassMemoryMgmt 59
in Storage management 41
cmmDefault
in ClassMemoryMgmt 59
in Storage management 41
cmmNonCICS
in ClassMemoryMgmt 59
in Storage management 41
Codes
in Enumerations 92
in IccCondition structure 92
col (parameter)
in send 226
in send3270Data 227
in sendLine 228
in setCursor 229
Color
in Enumerations 234
in IccTerminal class 234
color (parameter)
in operator« 221
in setColor 229
commArea
in IccControl class 98
commArea (parameter)
in link 151
in setNextCommArea 230
commitOnReturn
in CommitOpt 153
CommitOpt
in Enumerations 153
in IccProgram class 153
commitUOW
in IccTask class 201
compiling programs 30
Compiling Programs
in Compiling, executing, and debugging 30
complete
in Kind 144
complete key 15
completeLength
in IccKey class 141
in Public methods 141
completeLength (parameter)
in Constructor 141
condition

condition *(continued)*
in IccEvent class 109
in IccResource class 163
in Manual condition handling (noAction) 35
in Resource classes 9
condition (parameter)
in actionOnCondition 162
in actionOnConditionAsChar 162
in conditionText 57
in setActionOnCondition 165
condition 0 (NORMAL)
in actionsOnConditionsText 162
condition 1 (ERROR)
in actionsOnConditionsText 162
condition 2 (RDATT)
in actionsOnConditionsText 162
condition 3 (WRBRK)
in actionsOnConditionsText 162
condition 4 (ICCEOF)
in actionsOnConditionsText 162
condition 5 (EODS)
in actionsOnConditionsText 163
condition 6 (EOC)
in actionsOnConditionsText 163
Conditions, errors, and exceptions
Automatic condition handling (callHandleEvent) 35
Exception handling (throwException) 36
Manual condition handling (noAction) 35
Method level 37
Object level 37
Parameter level 37
Severe error handling (abendTask) 36
conditionText
in Functions 57
in Icc structure 57
in IccEvent class 109
in IccResource class 163
ConditionType
in Enumerations 167
in IccResource class 167
confirmation
in SendOpt 185
connectProcess
in IccSession class 176, 177
in Public methods 176, 177
console
in IccControl class 98
Constructor
in IccAbendData class 60
in IccAbendData constructor (protected) 60
in IccAbsTime class 65
in IccAbsTime constructor 65
in IccAlarmRequestId class 70
in IccAlarmRequestId constructors 70
in IccBase class 72
in IccBase constructor (protected) 72
in IccBuf class 76, 77
in IccBuf constructors 76, 77
in IccClock class 87
in IccClock constructor 87
in IccConsole class 94
in IccConsole constructor (protected) 94
in IccControl class 98
in IccControl constructor (protected) 98
Constructor *(continued)*
in IccConvId class 102
in IccConvId constructors 102
in IccDataQueue class 104
in IccDataQueue constructors 104
in IccDataQueueId class 107
in IccDataQueueId constructors 107
in IccEvent class 108
in IccEvent constructor 108
in IccException class 110
in IccException constructor 110
in IccFile class 114
in IccFile constructors 114
in IccFileId class 126
in IccFileId constructors 126
in IccFileIterator class 128
in IccFileIterator constructor 128
in IccGroupId class 131
in IccGroupId constructors 131
in IccJournal class 132, 133
in IccJournal constructors 132, 133
in IccJournalId class 137
in IccJournalId constructors 137
in IccJournalTypeId class 139
in IccJournalTypeId constructors 139
in IccKey class 141
in IccKey constructors 141
in IccLockId class 144, 145
in IccLockId constructors 144, 145
in IccMessage class 146
in IccMessage constructor 146
in IccPartnerId class 148
in IccPartnerId constructors 148
in IccProgram class 150
in IccProgram constructors 150
in IccProgramId class 154
in IccProgramId constructors 154
in IccRBA class 156
in IccRBA constructor 156
in IccRecordIndex class 158
in IccRecordIndex constructor (protected) 158
in IccRequestId class 159, 160
in IccRequestId constructors 159, 160
in IccResource class 161
in IccResource constructor (protected) 161
in IccResourceId class 168
in IccResourceId constructors (protected) 168
in IccRRN class 170
in IccRRN constructors 170
in IccSemaphore class 172
in IccSemaphore constructor 172
in IccSession class 175, 176
in IccSession constructor (protected) 176
in IccSession constructors (public) 175
in IccStartRequestQ class 186
in IccStartRequestQ constructor (protected) 186
in IccSysId class 192
in IccSysId constructors 192
in IccSystem class 194
in IccSystem constructor (protected) 194
in IccTask class 200
in IccTask Constructor (protected) 200
in IccTempStore class 211
in IccTempStore constructors 211

Constructor *(continued)*
in IccTempStoreId class 216
in IccTempStoreId constructors 216
in IccTermId class 217, 218
in IccTermId constructors 217, 218
in IccTerminal class 219
in IccTerminal constructor (protected) 219
in IccTerminalData class 235
in IccTerminalData constructor (protected) 235
in IccTime class 242
in IccTime constructor (protected) 242
in IccTimeInterval class 244, 245
in IccTimeInterval constructors 244, 245
in IccTimeOfDay class 247
in IccTimeOfDay constructors 247
in IccTPNameId class 249
in IccTPNameId constructors 249
in IccTransId class 250, 251
in IccTransId constructors 250, 251
in IccUser class 252
in IccUser constructors 252
in IccUserId class 256
in IccUserId constructors 256
converse
in IccSession class 177
convId
in IccSession class 178
convId (parameter)
in Constructor 103
convName (parameter)
in Constructor 102
in operator= 103
copt (parameter)
in setStartOpts 190
createDump
in AbendDumpOpt 209
creating a resource object 11
Creating a resource object
in Overview of the foundation classes 11
in Using CICS resources 11
Singleton classes 11
Creating an object
in C++ Objects 5
creating object 5
current (parameter)
in setPrefix 135
cursor
in Finding out information about a terminal 27
in IccTerminal class 220
customClassNum
in IccBase class 73
in Public methods 73
cut
in IccBuf class 79
in IccBuf constructors 13
CVDA
in Enumeration 257
in IccValue structure 257
cyan
in Color 234

D

data
data *(continued)*
in Accessing start data 21
in Finding out information about a terminal 27
in IccStartRequestQ class 187
in IccTerminal class 220
data (parameter)
in enterTrace 202
in put 181
data area extensibility 12
Data area extensibility
in Buffer objects 12
in IccBuf class 12
data area ownership 12
Data area ownership
in Buffer objects 12
in IccBuf class 12
dataArea
in IccBuf class 79
dataArea (parameter)
in append 78
in assign 78, 141
in Constructor 77
in insert 80
in overlay 85
in replace 85
dataAreaLength
in IccBuf class 79
in Public methods 79
dataAreaOwner
in Data area ownership 12
in IccBuf class 80
DataAreaOwner
in Enumerations 86
in IccBuf class 86
dataAreaType
in Data area extensibility 12
in IccBuf class 80
DataAreaType
in Enumerations 86
in IccBuf class 86
dataItems
in Example of polymorphic behavior 39
dataLength
in IccBuf class 80
dataqueue
in FacilityType 210
dataQueueTrigger
in StartType 210
date
in IccAbsTime class 66
in IccClock class 88
date services 28
dateFormat
in IccSystem class 195
DateFormat
in Enumerations 91
in IccClock class 91
dateSeparator (parameter)
in date 66, 88
in Example of time and date services 29
dayOfMonth
in Example of time and date services 29
in IccAbsTime class 66
in IccClock class 88

dayOfWeek
in Example of time and date services 29
in IccAbsTime class 66
in IccClock class 88
DayOfWeek
in Enumerations 91
in IccClock class 91
daysSince1900
in Example of time and date services 29
in IccAbsTime class 66
in IccClock class 89
daysUntilPasswordExpires
in IccUser class 253
dComplete
in DumpOpts 209
dDCT
in DumpOpts 209
dDefault
in DumpOpts 209
debugging programs 31
Debugging Programs
in Compiling, executing, and debugging 31
defaultColor
in Color 234
defaultHeight
in IccTerminalData class 236
in Public methods 236
defaultHighlight
in Highlight 234
defaultWidth
in IccTerminalData class 236
in Public methods 236
delay
in IccTask class 201
in Support Classes 10
deletable
in Access 125
delete
in Deleting an object 6
in Storage management 40, 41
delete operator 5
deleteLockedRecord
in Deleting locked records 17
in IccFile class 116
deleteRecord
in Deleting normal records 17
in IccFile class 116
deleteRecord method 17
Deleting an object
in C++ Objects 6
deleting items 26
Deleting items
in Temporary storage 26
in Using CICS Services 26
Deleting locked records
in Deleting records 17
in File control 17
Deleting normal records
in Deleting records 17
in File control 17
deleting queues 24
Deleting queues
in Transient Data 24
in Using CICS Services 24
deleting records 17
Deleting records
Deleting locked records 17
Deleting normal records 17
in File control 17
in Using CICS Services 17
dFCT
in DumpOpts 209
DFHCURDI 3
DFHCURDS 3
disabled
in Status 126
doSomething
in Using an object 6
dPCT
in DumpOpts 209
DPL
in StartType 210
dPPT
in DumpOpts 209
dProgram
in DumpOpts 209
dSIT
in DumpOpts 210
dStorage
in DumpOpts 209
dTables
in DumpOpts 209
dTask
in DumpOpts 209
dTCT
in DumpOpts 210
dTerminal
in DumpOpts 209
dTRT
in DumpOpts 210
dump
in IccTask class 201
dumpCode (parameter)
in dump 201, 202
DumpOpts
in Enumerations 209
in IccTask class 209
dynamic creation 5
dynamic deletion 5
dynamic link library 2
Dynamic link library
in Installed contents 2
Location 2

E

ECBList (parameter)
in waitExternal 207
EDF (parameter)
in initializeEnvironment 57
empty
in Deleting items 26
in Deleting queues 24
in IccDataQueue class 104
in IccTempStore class 212
in Temporary storage 25
in Transient Data 23
enabled

enabled *(continued)*
in Status 126
enableStatus
in IccFile class 116
endBrowse
in IccSystem class 195
endInsert
in Writing records 16
endInsert (VSAM only)
in IccFile class 117
in Public methods 117
endl
in Example of terminal control 28
ENTER
in AIDVal 234
enterTrace
in IccTask class 202
entryPoint
in IccProgram class 151
Enumeration
CVDA 257
in IccValue structure 257
Enumerations
AbendDumpOpt 209
AbendHandlerOpt 209
Access 125
ActionOnCondition 167
AIDVal 234
AllocateOpt 185
Bool 58
BoolSet 59
Case 234
CheckOpt 192
ClassMemoryMgmt 59
ClassType 75
Codes 92
Color 234
CommitOpt 153
ConditionType 167
DataAreaOwner 86
DataAreaType 86
DateFormat 91
DayOfWeek 91
DumpOpts 209
FacilityType 210
FamilySubset 59
GetOpt 60
HandleEventReturnOpt 167
Highlight 234
in Icc structure 58
in IccBase class 75
in IccBuf class 86
in IccClock class 91
in IccCondition structure 92
in IccConsole class 97
in IccException class 113
in IccFile class 125
in IccJournal class 136
in IccKey class 144
in IccProgram class 153
in IccRecordIndex class 159
in IccResource class 167
in IccSemaphore class 174
in IccSession class 185
Enumerations *(continued)*
in IccStartRequestQ class 192
in IccSystem class 199
in IccTask class 209
in IccTempStore class 215
in IccTerminal class 234
in IccTime class 244
Kind 144
LifeTime 174
LoadOpt 154
Location 215
LockType 174
MonthOfYear 92
NameOpt 76
NextTransIdOpt 235
NoSpaceOpt 215
Options 136
Platforms 60
ProtectOpt 192
Range 93
ReadMode 125
ResourceType 199
RetrieveOpt 192
SearchCriterion 126
SendOpt 185
SeverityOpt 97
StartType 210
StateOpt 185
Status 126
StorageOpts 210
SyncLevel 185
TraceOpt 210
Type 113, 159, 244
UpdateMode 92
WaitPostType 211
WaitPurgeability 211
equalToKey
in SearchCriterion 126
erase
in Example of terminal control 28
in IccTerminal class 220
in Sending data to a terminal 27
errorCode
in IccSession class 178
ESDS
in File control 15
ESDS file 15
ESMReason
in IccUser class 253
ESMResponse
in IccUser class 253
event (parameter)
in handleEvent 164
Example of file control
in File control 17
in Using CICS Services 17
Example of managing transient data
in Transient Data 24
in Using CICS Services 24
Example of polymorphic behavior
in Miscellaneous 39
in Polymorphic Behavior 39
Example of starting transactions
in Starting transactions asynchronously 21

Example of starting transactions *(continued)*
in Using CICS Services 21
Example of Temporary Storage
in Temporary storage 26
in Using CICS Services 26
Example of terminal control
in Terminal control 27
in Using CICS Services 27
Example of time and date services
in Time and date services 29
in Using CICS Services 29
exception
in TraceOpt 210
exception (parameter)
in catchException 56
Exception handling (throwException)
in CICS conditions 36
in Conditions, errors, and exceptions 36
exceptionNum (parameter)
in Constructor 111
exceptions 32
exceptionType (parameter)
in Constructor 111
Executing Programs
in Compiling, executing, and debugging 31
extensible
in DataAreaType 87
external
in DataAreaOwner 86
extractProcess
in IccSession class 178
extractState
in StateOpt 185

F

facilityType
in IccTask class 202
FacilityType
in Enumerations 210
in IccTask class 210
fam (parameter)
in initializeEnvironment 57
familyConformanceError
in C++ Exceptions and the Foundation Classes 34
in Type 113
FamilySubset
in Enumerations 59
in Icc structure 59
FEPIRequest
in StartType 210
file (parameter)
in Constructor 128
in Example of file control 18
file control
browsing records 17
deleting records 17
example 17
rewriting records 17
updating records 17
File control
Browsing records 17
Deleting locked records 17
Deleting normal records 17
File control *(continued)*
Deleting records 17
Example of file control 17
in Using CICS Services 15
Reading ESDS records 16
Reading KSDS records 15
Reading records 15
Reading RRDS records 16
Updating records 17
Writing ESDS records 16
Writing KSDS records 16
Writing records 16
Writing RRDS records 16
fileName (parameter)
in Constructor 114, 126
in getFile 196
in operator= 127
Finding out information about a terminal
in Terminal control 27
in Using CICS Services 27
fixed
in DataAreaType 87
flush
in Example of terminal control 28
in IccSession class 178
for
in Example of file control 18
Form
in Polymorphic Behavior 39
format (parameter)
in append 78
in assign 79
in date 66, 88
in Example of time and date services 29
in send 225, 226
in send3270Data 227
in sendLine 228
Foundation Class Abend codes
in Conditions, errors, and exceptions 32
free
in IccSession class 178
freeKeyboard
in IccTerminal class 220
in Sending data to a terminal 27
freeStorage
in IccSystem class 195
in IccTask class 202
fsAllowPlatformVariance
in FamilySubset 59
in Platform differences 36
fsDefault
in FamilySubset 59
fsEnforce
in FamilySubset 59
in Platform differences 36
fullAccess
in Access 125
Functions
boolText 56
catchException 56
conditionText 57
in Icc structure 56
initializeEnvironment 57
isClassMemoryMgmtOn 57

Functions *(continued)*
isEDFOn 57
isFamilySubsetEnforcementOn 58
returnToCICS 58
setEDF 58
unknownException 58

G

generic
in Kind 144
generic key 15
get
in Example of polymorphic behavior 40
in IccDataQueue class 105
in IccResource class 163
in IccSession class 179
in IccTempStore class 212
in IccTerminal class 220
in Polymorphic Behavior 39
getFile
in IccSystem class 196
getNextFile
in IccSystem class 196
GetOpt
in Enumerations 60
in Icc structure 60
getStorage
in IccSystem class 196
in IccTask class 203
gid (parameter)
in Constructor 252
graphicCharCodeSet
in IccTerminalData class 236
graphicCharSetId
in IccTerminalData class 237
green
in Color 234
groupId
in IccUser class 253
groupName (parameter)
in Constructor 131, 252
in operator= 131
gteqToKey
in SearchCriterion 126

H

H
in actionOnConditionAsChar 162
handleEvent
in Automatic condition handling (callHandleEvent) 35
in IccResource class 163
HandleEventReturnOpt
in Enumerations 167
in IccResource class 167
handPost
in WaitPostType 211
Header files
in Installed contents 1, 30
Location 2
height
in IccTerminal class 220
Highlight
in Enumerations 234
in IccTerminal class 234
highlight (parameter)
in operator« 221
in setHighlight 229, 230
hold
in LoadOpt 154
hours
in IccAbsTime class 67
in IccTime class 242
hours (parameter)
in Constructor 242, 244, 245, 247
in set 245, 247

I

Icc
in Foundation Classes—reference 43
in Foundation Classes: reference 43
in Method level 37
in Overview of the foundation classes 6
Icc structure
Bool 58
BoolSet 59
boolText 56
catchException 56
ClassMemoryMgmt 59
conditionText 57
FamilySubset 59
GetOpt 60
initializeEnvironment 57
isClassMemoryMgmtOn 57
isEDFOn 57
isFamilySubsetEnforcementOn 58
Platforms 60
returnToCICS 58
setEDF 58
unknownException 58
Icc::initializeEnvironment
in Storage management 40
ICC$BUF 3
ICC$CLK 3
ICC$HEL 3
ICC$SES1 3
ICC$SES2 3
IccAbendData
in Singleton classes 11
IccAbendData class
abendCode 60
ASRAInterrupt 61
ASRAKeyType 61
ASRAPSW 62
ASRARegisters 62
ASRASpaceType 62
ASRAStorageType 63
Constructor 60
instance 63
isDumpAvailable 63
originalAbendCode 64
programName 64
IccAbendData constructor (protected)
Constructor 60
in IccAbendData class 60

IccAbsTime
in Base classes 7
in delay 201
in IccTime class 242
in Support Classes 10
in Time and date services 28
IccAbsTime class
Constructor 65
date 66
dayOfMonth 66
dayOfWeek 66
daysSince1900 66
hours 67
milliSeconds 67
minutes 67
monthOfYear 67
operator= 67
packedDecimal 67
seconds 68
time 68
timeInHours 68
timeInMinutes 68
timeInSeconds 68
year 68
IccAbsTime constructor
Constructor 65
in IccAbsTime class 65
IccAbsTime,
in Support Classes 10
IccAlarmRequestId
in IccAlarmRequestId class 70
IccAlarmRequestId class
Constructor 70
isExpired 71
operator= 71
setTimerECA 71
timerECA 72
IccAlarmRequestId constructors
Constructor 70
in IccAlarmRequestId class 70
IccBase
in Base classes 6
in Foundation Classes—reference 43
in IccAbendData class 60
in IccAbsTime class 65
in IccAlarmRequestId class 70
in IccBase class 72
in IccBuf class 76
in IccClock class 87
in IccConsole class 93
in IccControl class 97
in IccConvId class 102
in IccDataQueue class 104
in IccDataQueueId class 107
in IccEvent class 108
in IccException class 110
in IccFile class 114
in IccFileId class 126
in IccFileIterator class 128
in IccGroupId class 131
in IccJournal class 132
in IccJournalId class 137
in IccJournalTypeId class 139
IccBase *(continued)*
in IccKey class 140
in IccLockId class 144
in IccMessage class 146
in IccPartnerId class 148
in IccProgram class 150
in IccProgramId class 154
in IccRBA class 155
in IccRecordIndex class 158
in IccRequestId class 159
in IccResource class 161
in IccResourceId class 168
in IccRRN class 169
in IccSemaphore class 172
in IccSession class 175
in IccStartRequestQ class 186
in IccSysId class 192
in IccSystem class 194
in IccTask class 200
in IccTempStore class 211
in IccTempStoreId class 216
in IccTermId class 217
in IccTerminal class 219
in IccTerminalData class 235
in IccTime class 242
in IccTimeInterval class 244
in IccTimeOfDay class 246
in IccTPNameId class 248
in IccTransId class 250
in IccUser class 252
in IccUserId class 256
in Resource classes 8
in Resource identification classes 7
in Storage management 40, 41
in Support Classes 10
IccBase class
className 73
classType 73
ClassType 75
Constructor 72
customClassNum 73
NameOpt 76
operator delete 73
operator new 74
overview 6
setClassName 74
setCustomClassNum 74
IccBase constructor (protected)
Constructor 72
in IccBase class 72
IccBuf
in Buffer objects 12
in C++ Exceptions and the Foundation Classes 34
in Data area extensibility 12, 13
in Data area ownership 12
in Example of file control 18
in Example of managing transient data 25
in Example of polymorphic behavior 39
in Example of starting transactions 22, 23
in Example of Temporary Storage 26
in Example of terminal control 28
in IccBuf class 12, 76
in IccBuf constructors 13
in IccBuf methods 14

IccBuf *(continued)*
in Reading data 24
in Reading items 25
in Scope of data in IccBuf reference returned from 'read'
methods 41, 42
in Support Classes 11
in Working with IccResource subclasses 14
IccBuf class
append 78
assign 78, 79
Constructor 76, 77
constructors 13
cut 79
data area extensibility 12
Data area extensibility 12
data area ownership 12
Data area ownership 12
dataArea 79
dataAreaLength 79
dataAreaOwner 80
DataAreaOwner 86
dataAreaType 80
DataAreaType 86
dataLength 80
IccBuf constructors 13
IccBuf methods 14
in Buffer objects 12
insert 80
isFMHContained 80
methods 14
operator const char* 81
operator!= 82
operator« 82, 84
operator+= 81, 82
operator= 81
operator== 82
overlay 85
replace 85
setDataLength 85
setFMHContained 86
Working with IccResource subclasses 14
IccBuf constructors
Constructor 76, 77
in Buffer objects 13
in IccBuf class 13, 76
IccBuf methods
in Buffer objects 14
in IccBuf class 14
IccBuf reference 41
IccClock
in Example of time and date services 29
in IccAlarmRequestId class 70
in IccClock class 87
in Time and date services 28
IccClock class
absTime 87
cancelAlarm 87
Constructor 87
date 88
DateFormat 91
dayOfMonth 88
dayOfWeek 88
DayOfWeek 91
daysSince1900 89
IccClock class *(continued)*
milliSeconds 89
monthOfYear 89
MonthOfYear 92
setAlarm 89
time 90
update 90
UpdateMode 92
year 90
IccClock constructor
Constructor 87
in IccClock class 87
IccCondition
in C++ Exceptions and the Foundation Classes 34
IccCondition structure
Codes 92
Range 93
IccConsole
in Buffer objects 12
in Object level 37
in Singleton classes 11
IccConsole class
Constructor 94
instance 94
overview 11
put 94
replyTimeout 94
resetRouteCodes 94
setAllRouteCodes 95
setReplyTimeout 95
setRouteCodes 95
SeverityOpt 97
write 96
writeAndGetReply 96
IccConsole constructor (protected)
Constructor 94
in IccConsole class 94
IccControl
in Base classes 7
in Example of starting transactions 22, 23
in IccControl class 97
in IccProgram class 150
in main function 258
in Mapping EXEC CICS calls to Foundation Class
methods 44
in Method level 37
in Singleton classes 11
in Support Classes 10
IccControl class
callingProgramId 98
cancelAbendHandler 98
commArea 98
console 98
Constructor 98
initData 99
instance 99
isCreated 99
overview 7, 11
programId 99
resetAbendHandler 99
returnProgramId 100
run 100
session 100
setAbendHandler 100

IccControl class *(continued)*
startRequestQ 101
system 101
task 101
terminal 101
IccControl constructor (protected)
Constructor 98
in IccControl class 97
IccControl::run
in Mapping EXEC CICS calls to Foundation Class methods 44
IccConvId
in IccConvId class 102
IccConvId class
Constructor 102
operator= 103
IccConvId constructors
Constructor 102
in IccConvId class 102
IccDataQueue
in Buffer objects 12
in Example of managing transient data 24
in Example of polymorphic behavior 40
in Resource classes 9
in Temporary storage 25
in Transient Data 23, 24
in Working with IccResource subclasses 14
in Writing data 24
IccDataQueue class
clear 104
Constructor 104
empty 104
get 105
put 105
readItem 105
writeItem 105
IccDataQueue constructors
Constructor 104
in IccDataQueue class 104
IccDataQueueId
in Example of managing transient data 24
in IccDataQueueId class 107
in Transient Data 23, 24
IccDataQueueId class
Constructor 107
operator= 107
IccDataQueueId constructors
Constructor 107
in IccDataQueueId class 107
IccEvent
in IccEvent class 108
in Support Classes 11
IccEvent class
className 109
classType 109
condition 109
conditionText 109
Constructor 108
methodName 109
summary 110
IccEvent constructor
Constructor 108
in IccEvent class 108
IccException
IccException *(continued)*
in C++ Exceptions and the Foundation Classes 33, 34
in IccException class 110
in IccMessage class 146
in main function 259
in Method level 37
in Object level 37
in Parameter level 38
in Support Classes 11
IccException class
CICSCondition type 34
className 111
classType 111
Constructor 110
familyConformanceError type 34
internalError type 34
invalidArgument type 34
invalidMethodCall type 34
message 111
methodName 112
number 112
objectCreationError type 34
summary 112
type 112
Type 113
typeText 112
IccException constructor
Constructor 110
in IccException class 110
ICCFCC 3
ICCFCCL 3
ICCFCGL 3
ICCFCIMP 3
ICCFCL 3
IccFile
in Browsing records 17
in Buffer objects 12
in C++ Exceptions and the Foundation Classes 34
in Deleting locked records 17
in Deleting normal records 17
in Example of file control 17
in File control 15
in IccFile class 114
in IccFileIterator class 128
in Reading ESDS records 16
in Reading KSDS records 15
in Reading records 15
in Reading RRDS records 16
in Resource identification classes 7
in Singleton classes 11
in Updating records 17
in Writing ESDS records 16
in Writing KSDS records 16
in Writing records 16
in Writing RRDS records 17
IccFile class
access 115
Access 125
accessMethod 115
beginInsert (VSAM only) 115
Constructor 114
deleteLockedRecord 17, 116
deleteRecord 116
deleteRecord method 17

IccFile class *(continued)*
enableStatus 116
endInsert (VSAM only) 117
isAddable 117
isBrowsable 117
isDeletable 117
isEmptyOnOpen 118
isReadable 118
isReadable method 15, 16
isRecoverable 118
isUpdatable 119
keyLength 119
keyLength method 15
keyPosition 119
keyPosition method 15
openStatus 119
ReadMode 125
readRecord 120
readRecord method 15
recordFormat 120
recordFormat method 16
recordIndex 121
recordIndex method 15, 16
recordLength 121
recordLength method 15, 16
registerRecordIndex 16, 121
registerRecordIndex method 15
rewriteRecord 122
rewriteRecord method 17
SearchCriterion 126
setAccess 122
setEmptyOnOpen 122
setStatus 123
Status 126
type 123
unlockRecord 123
writeRecord 124
writeRecord method 16
IccFile constructors
Constructor 114
in IccFile class 114
IccFile::readRecord
in Scope of data in IccBuf reference returned from 'read' methods 41
IccFileId
in Base classes 7
in File control 15
in IccFileId class 126
in Resource identification classes 7
IccFileId class
Constructor 126
operator= 127
overview 7, 15
reading records 15
IccFileId constructors
Constructor 126
in IccFileId class 126
IccFileIterator
in Browsing records 17
in Buffer objects 12
in Example of file control 17, 18
in File control 15
in IccFileIterator class 128
IccFileIterator class
IccFileIterator class *(continued)*
Constructor 128
overview 15
readNextRecord 129
readNextRecord method 17
readPreviousRecord 17, 129
reset 129
IccFileIterator constructor
Constructor 128
in IccFileIterator class 128
IccGroupId
in IccGroupId class 131
IccGroupId class
Constructor 131
operator= 131, 132
IccGroupId constructors
Constructor 131
in IccGroupId class 131
IccJournal
in Buffer objects 12
in IccJournal class 132
in Object level 37
IccJournal class
clearPrefix 133
Constructor 132, 133
journalTypeId 133
Options 136
put 134
registerPrefix 134
setJournalTypeId 134
setPrefix 134
wait 135
writeRecord 135
IccJournal constructors
Constructor 132, 133
in IccJournal class 132
IccJournalId
in IccJournalId class 137
IccJournalId class
Constructor 137
number 138
operator= 138
IccJournalId constructors
Constructor 137
in IccJournalId class 137
IccJournalTypeId
in Foundation Classes—reference 43
in IccJournalTypeId class 139
IccJournalTypeId class
Constructor 139
operator= 139, 140
IccJournalTypeId constructors
Constructor 139
in IccJournalTypeId class 139
IccKey
in Browsing records 17
in Deleting normal records 17
in File control 15
in IccKey class 140
in IccRecordIndex class 158
in Reading KSDS records 15
in Reading records 15
in Writing KSDS records 16

IccKey *(continued)*
in Writing records 16
IccKey class
assign 141
completeLength 141
Constructor 141
kind 142
Kind 144
operator!= 143
operator= 142
operator== 142
reading records 15
setKind 143
value 143
IccKey constructors
Constructor 141
in IccKey class 141
IccLockId
in IccLockId class 144
IccLockId class
Constructor 144, 145
operator= 145
IccLockId constructors
Constructor 144, 145
in IccLockId class 144
IccMessage
in IccMessage class 146
in Support Classes 11
IccMessage class
className 147
Constructor 146
methodName 147
number 147
summary 147
text 147
IccMessage constructor
Constructor 146
in IccMessage class 146
IccPartnerId
in IccPartnerId class 148
IccPartnerId class
Constructor 148
operator= 149
IccPartnerId constructors
Constructor 148
in IccPartnerId class 148
IccProgram
in Buffer objects 12
in IccProgram class 150
in Program control 19
in Resource classes 9
IccProgram class
address 150
clearInputMessage 151
CommitOpt 153
Constructor 150
entryPoint 151
length 151
link 151
load 152
LoadOpt 154
program control 19
setInputMessage 152
unload 152
IccProgram constructors
Constructor 150
in IccProgram class 150
IccProgramId
in IccProgramId class 154
in Resource identification classes 7
IccProgramId class
Constructor 154
operator= 154, 155
IccProgramId constructors
Constructor 154
in IccProgramId class 154
IccRBA
in Browsing records 17
in File control 15
in IccRBA class 155
in IccRecordIndex class 158
in Reading ESDS records 16
in Reading records 15
in Writing ESDS records 16
in Writing records 16
in Writing RRDS records 16
IccRBA class
Constructor 156
number 157
operator!= 157
operator= 156
operator== 156
reading records 15
IccRBA constructor
Constructor 156
in IccRBA class 155
IccRecordIndex
in C++ Exceptions and the Foundation Classes 34
in IccRecordIndex class 158
IccRecordIndex class
Constructor 158
length 158
type 158
Type 159
IccRecordIndex constructor (protected)
Constructor 158
in IccRecordIndex class 158
IccRequestId
in Example of starting transactions 21, 22
in IccRequestId class 159
in Parameter passing conventions 41
IccRequestId class
Constructor 159, 160
operator= 160
IccRequestId constructors
Constructor 159, 160
in IccRequestId class 159
IccResource
in Base classes 6, 7
in Example of polymorphic behavior 39, 40
in IccResource class 161
in Polymorphic Behavior 39
in Resource classes 9
in Scope of data in IccBuf reference returned from 'read' methods 41, 42
IccResource class
actionOnCondition 162
ActionOnCondition 167

IccResource class *(continued)*
actionOnConditionAsChar 162
actionsOnConditionsText 163
clear 163
condition 163
conditionText 163
ConditionType 167
Constructor 161
get 163
handleEvent 163
HandleEventReturnOpt 167
id 164
isEDFOn 164
isRouteOptionOn 164
name 164
overview 6, 7
put 164
routeOption 165
setActionOnAnyCondition 165
setActionOnCondition 165
setActionsOnConditions 165
setEDF 166
setRouteOption 166
working with subclasses 14
IccResource constructor (protected)
Constructor 161
in IccResource class 161
IccResourceId
in Base classes 6, 7
in C++ Exceptions and the Foundation Classes 34
in Resource identification classes 7
IccResourceId class
Constructor 168
name 168
nameLength 169
operator= 169
overview 6, 7
IccResourceId constructors (protected)
Constructor 168
in IccResourceId class 168
IccRRN
in Browsing records 17
in Deleting normal records 17
in File control 15
in IccRecordIndex class 158
in IccRRN class 169
in Reading records 15
in Reading RRDS records 16
in Writing records 16
IccRRN class
Constructor 170
number 171
operator!= 171
operator= 170
operator== 170
reading records 15
IccRRN constructors
Constructor 170
in IccRRN class 170
IccSemaphore class
Constructor 172
lifeTime 173
LifeTime 174
lock 173
IccSemaphore class *(continued)*
LockType 174
tryLock 173
type 173
unlock 173
IccSemaphore constructor
Constructor 172
in IccSemaphore class 172
IccSession
in Buffer objects 12
IccSession class
allocate 176
AllocateOpt 185
connectProcess 176, 177
Constructor 175, 176
converse 177
convId 178
errorCode 178
extractProcess 178
flush 178
free 178
get 179
isErrorSet 179
isNoDataSet 179
isSignalSet 179
issueAbend 179
issueConfirmation 179
issueError 180
issuePrepare 180
issueSignal 180
PIPList 180
process 180
put 181
receive 181
send 181
sendInvite 182
sendLast 182, 183
SendOpt 185
state 183
StateOpt 185
stateText 184
syncLevel 184
SyncLevel 185
IccSession constructor (protected)
Constructor 176
in IccSession class 176
IccSession constructors (public)
Constructor 175
in IccSession class 175
IccStartRequestQ
in Accessing start data 21
in Buffer objects 12
in Example of starting transactions 22, 23
in IccRequestId class 159
in IccStartRequestQ class 186
in Mapping EXEC CICS calls to Foundation Class methods 44
in Parameter passing conventions 41
in Singleton classes 11
in Starting transactions asynchronously 20
IccStartRequestQ class
cancel 186
CheckOpt 192
clearData 187

IccStartRequestQ class *(continued)*
Constructor 186
data 187
instance 187
overview 11
ProtectOpt 192
queueName 187
registerData 187
reset 188
retrieveData 188
RetrieveOpt 192
returnTermId 188
returnTransId 188
setData 188
setQueueName 189
setReturnTermId 189
setReturnTransId 189, 190
setStartOpts 190
start 190
IccStartRequestQ constructor (protected)
Constructor 186
in IccStartRequestQ class 186
IccSysId
in IccSysId class 192
in Program control 19
IccSysId class
Constructor 192
operator= 193
IccSysId constructors
Constructor 192
in IccSysId class 192
IccSystem
in Singleton classes 11
IccSystem class
applName 194
beginBrowse 194, 195
Constructor 194
dateFormat 195
endBrowse 195
freeStorage 195
getFile 196
getNextFile 196
getStorage 196
instance 197
operatingSystem 197
operatingSystemLevel 197
overview 11
release 197
releaseText 198
ResourceType 199
sysId 198
workArea 198
IccSystem constructor (protected)
Constructor 194
in IccSystem class 194
IccTask
in C++ Exceptions and the Foundation Classes 34
in Example of starting transactions 23
in IccAlarmRequestId class 70
in IccTask class 200
in Parameter level 37
in Singleton classes 11
in Support Classes 10
IccTask class
IccTask class *(continued)*
abend 200
abendData 200
AbendDumpOpt 209
AbendHandlerOpt 209
commitUOW 201
Constructor 200
delay 201
dump 201
DumpOpts 209
enterTrace 202
facilityType 202
FacilityType 210
freeStorage 202
getStorage 203
instance 203
isCommandSecurityOn 203
isCommitSupported 204
isResourceSecurityOn 204
isRestarted 204
isStartDataAvailable 204
number 204
overview 11
principalSysId 205
priority 205
rollBackUOW 205
setDumpOpts 205
setPriority 206
setWaitText 206
startType 206
StartType 210
StorageOpts 210
suspend 206
TraceOpt 210
transId 206
triggerDataQueueId 206
userId 207
waitExternal 207
waitOnAlarm 207
WaitPostType 211
WaitPurgeability 211
workArea 208
IccTask Constructor (protected)
Constructor 200
in IccTask class 200
IccTask::commitUOW
in Scope of data in IccBuf reference returned from 'read' methods 42
IccTempstore
in Working with IccResource subclasses 14
IccTempStore
in Automatic condition handling (callHandleEvent) 35
in Buffer objects 12
in C++ Exceptions and the Foundation Classes 34
in Deleting items 26
in Example of polymorphic behavior 40
in Example of Temporary Storage 26
in IccTempStore class 211
in Reading items 25
in Resource classes 9
in Temporary storage 25
in Transient Data 24
in Updating items 25
in Working with IccResource subclasses 14

IccTempStore *(continued)*
in Writing items 25
IccTempStore class
clear 212
Constructor 211
empty 212
get 212
Location 215
NoSpaceOpt 215
numberOfItems 212
put 212
readItem 213
readNextItem 213
rewriteItem 213
writeItem 214
IccTempStore constructors
Constructor 211
in IccTempStore class 211
IccTempStore::readItem
in Scope of data in IccBuf reference returned from 'read' methods 41
IccTempStore::readNextItem
in Scope of data in IccBuf reference returned from 'read' methods 42
IccTempStoreId
in Base classes 7
in Example of Temporary Storage 26
in IccTempStoreId class 216
in Temporary storage 25
IccTempStoreId class
Constructor 216
operator= 216
IccTempStoreId constructors
Constructor 216
in IccTempStoreId class 216
IccTermId
in Base classes 6
in C++ Exceptions and the Foundation Classes 34
in Example of starting transactions 22
in Example of terminal control 27
in IccTermId class 217
in Terminal control 27
IccTermId class
Constructor 217, 218
operator= 218
overview 6
IccTermId constructors
Constructor 217, 218
in IccTermId class 217
IccTerminal
in Buffer objects 12
in Example of terminal control 27
in Finding out information about a terminal 27
in IccTerminalData class 235
in Receiving data from a terminal 27
in Resource classes 8, 9
in Singleton classes 11
in Terminal control 27
IccTerminal class
AID 219
AIDVal 234
Case 234
clear 219
Color 234
IccTerminal class *(continued)*
Constructor 219
cursor 220
data 220
erase 220
freeKeyboard 220
get 220
height 220
Highlight 234
inputCursor 221
instance 221
line 221
netName 221
NextTransIdOpt 235
operator« 221–224
put 224
receive 224
receive3270Data 225
registerInputMessage 152
send 225, 226
send3270Data 226, 227
sendLine 227, 228
setColor 229
setCursor 229
setHighlight 229
setLine 230
setNewLine 230
setNextCommArea 230
setNextInputMessage 230
setNextTransId 231
signoff 231
signon 231, 232
waitForAID 232
width 232
workArea 233
IccTerminal constructor (protected)
Constructor 219
in IccTerminal class 219
IccTerminal::receive
in Scope of data in IccBuf reference returned from 'read' methods 41
IccTerminalData
in Example of terminal control 27
in Finding out information about a terminal 27
in IccTerminalData class 235
in Terminal control 27
IccTerminalData class
alternateHeight 235
alternateWidth 236
Constructor 235
defaultHeight 236
defaultWidth 236
graphicCharCodeSet 236
graphicCharSetId 237
isAPLKeyboard 237
isAPLText 237
isBTrans 237
isColor 238
isEWA 238
isExtended3270 238
isFieldOutline 238
isGoodMorning 239
isHighlight 239
isKatakana 239

IccTerminalData class *(continued)*
isMSRControl 239
isPS 240
isSOSI 240
isTextKeyboard 240
isTextPrint 240
isValidation 241
IccTerminalData constructor (protected)
Constructor 235
in IccTerminalData class 235
IccTime
in Base classes 7
in IccTime class 242
in Parameter passing conventions 41
in Support Classes 10
IccTime class
Constructor 242
hours 242
minutes 242
overview 7
seconds 242
timeInHours 243
timeInMinutes 243
timeInSeconds 243
type 243
Type 244
IccTime constructor (protected)
Constructor 242
in IccTime class 242
IccTimeInterval
in Base classes 7
in delay 201
in Example of starting transactions 22, 23
in IccTime class 242
in Support Classes 10
IccTimeInterval class
Constructor 244, 245
operator= 245
set 245
IccTimeInterval constructors
Constructor 244, 245
in IccTimeInterval class 244
IccTimeOfDay
in Base classes 7
in delay 201
in IccTime class 242
in Support Classes 10
IccTimeOfDay class
Constructor 247
operator= 247
set 247
IccTimeOfDay constructors
Constructor 247
in IccTimeOfDay class 246
IccTPNameId
in IccTPNameId class 249
IccTPNameId class
Constructor 249
operator= 249
IccTPNameId constructors
Constructor 249
in IccTPNameId class 249
IccTransId
in Base classes 6
IccTransId *(continued)*
in Example of starting transactions 22
in IccResourceId class 168
in IccTransId class 250
in Parameter passing conventions 41
IccTransId class
Constructor 250, 251
operator= 251
overview 6
IccTransId constructors
Constructor 250, 251
in IccTransId class 250
IccUser class
changePassword 253
Constructor 252
daysUntilPasswordExpires 253
ESMReason 253
ESMResponse 253
groupId 253
invalidPasswordAttempts 254
language 254
lastPasswordChange 254
lastUseTime 254
passwordExpiration 254
setLanguage 254
verifyPassword 254
IccUser constructors
Constructor 252
in IccUser class 252
IccUserControl
in C++ Exceptions and the Foundation Classes 33
in Example of file control 18
in Example of managing transient data 24
in Example of polymorphic behavior 39
in Example of starting transactions 21
in Example of Temporary Storage 26
in Example of terminal control 28
in Example of time and date services 29
in main function 258
in Program control 19
in Singleton classes 11
IccUserId
in IccUserId class 256
IccUserId class
Constructor 256
operator= 256, 257
IccUserId constructors
Constructor 256
in IccUserId class 256
IccValue
in Foundation Classes: reference 43
IccValue structure
CVDA 257
id
in IccResource class 164
Id
in Resource identification classes 7
id (parameter)
in Constructor 70, 104, 107, 114, 127, 131, 133, 137, 139, 145, 148, 150, 154, 160, 168, 172, 175, 193, 211, 216, 218, 249, 251, 252, 256
in getFile 196
in operator= 71, 103, 108, 127, 132, 138, 140, 145, 149, 155, 160, 169, 193, 217, 218, 249–251, 257

id (parameter) *(continued)*
in setJournalTypeId 134
in signon 231
in waitOnAlarm 208
ifSOSReturnCondition
in StorageOpts 210
ignoreAbendHandler
in AbendHandlerOpt 209
immediate
in NextTransIdOpt 235
index (parameter)
in Constructor 114, 128
in registerRecordIndex 121
in reset 130
Inherited protected methods
in IccAbendData class 65
in IccAbsTime class 69
in IccAlarmRequestId class 72
in IccBuf class 86
in IccClock class 91
in IccConsole class 97
in IccControl class 102
in IccConvId class 103
in IccDataQueue class 106
in IccDataQueueId class 108
in IccEvent class 110
in IccException class 113
in IccFile class 125
in IccFileId class 128
in IccFileIterator class 130
in IccGroupId class 132
in IccJournal class 136
in IccJournalId class 139
in IccJournalTypeId class 140
in IccKey class 144
in IccLockId class 146
in IccMessage class 148
in IccPartnerId class 149
in IccProgram class 153
in IccProgramId class 155
in IccRBA class 157
in IccRecordIndex class 159
in IccRequestId class 161
in IccResource class 167
in IccResourceId class 169
in IccRRN class 171
in IccSemaphore class 174
in IccSession class 185
in IccStartRequestQ class 192
in IccSysId class 193
in IccSystem class 199
in IccTask class 209
in IccTempStore class 215
in IccTempStoreId class 217
in IccTermId class 219
in IccTerminal class 233
in IccTerminalData class 241
in IccTime class 244
in IccTimeInterval class 246
in IccTimeOfDay class 248
in IccTPNameId class 250
in IccTransId class 252
in IccUser class 255
in IccUserId class 257
Inherited public methods
in IccAbendData class 64
in IccAbsTime class 69
in IccAlarmRequestId class 72
in IccBuf class 86
in IccClock class 90
in IccConsole class 96
in IccControl class 101
in IccConvId class 103
in IccDataQueue class 106
in IccDataQueueId class 108
in IccEvent class 110
in IccException class 113
in IccFile class 124
in IccFileId class 127
in IccFileIterator class 130
in IccGroupId class 132
in IccJournal class 136
in IccJournalId class 138
in IccJournalTypeId class 140
in IccKey class 144
in IccLockId class 145
in IccMessage class 147
in IccPartnerId class 149
in IccProgram class 153
in IccProgramId class 155
in IccRBA class 157
in IccRecordIndex class 159
in IccRequestId class 161
in IccResource class 167
in IccResourceId class 169
in IccRRN class 171
in IccSemaphore class 174
in IccSession class 184
in IccStartRequestQ class 191
in IccSysId class 193
in IccSystem class 198
in IccTask class 208
in IccTempStore class 214
in IccTempStoreId class 217
in IccTermId class 218
in IccTerminal class 233
in IccTerminalData class 241
in IccTime class 243
in IccTimeInterval class 245
in IccTimeOfDay class 248
in IccTPNameId class 250
in IccTransId class 251
in IccUser class 255
in IccUserId class 257
initByte (parameter)
in getStorage 196, 197, 203
initData
in IccControl class 99
in Public methods 99
initializeEnvironment
in Functions 57
in Icc structure 57
in Method level 37
in Storage management 40, 41
initRBA (parameter)
in Constructor 156
initRRN (parameter)
in Constructor 170

initValue (parameter)
in Constructor 141
inputCursor
in IccTerminal class 221
insert
in Example of Temporary Storage 26
in IccBuf class 80
in IccBuf constructors 13
Installed contents
Location 2
instance
in IccAbendData class 63
in IccConsole class 94
in IccControl class 99
in IccStartRequestQ class 187
in IccSystem class 197
in IccTask class 203
in IccTerminal class 221
in Singleton classes 12
internal
in DataAreaOwner 86
internalError
in C++ Exceptions and the Foundation Classes 34
in Type 113
interval (parameter)
in setReplyTimeout 95
invalidArgument
in C++ Exceptions and the Foundation Classes 34
in Type 113
invalidMethodCall
in C++ Exceptions and the Foundation Classes 34
in Type 113
invalidPasswordAttempts
in IccUser class 254
isAddable
in IccFile class 117
in Writing ESDS records 16
in Writing KSDS records 16
in Writing RRDS records 17
isAPLKeyboard
in IccTerminalData class 237
in Public methods 237
isAPLText
in IccTerminalData class 237
in Public methods 237
isBrowsable
in IccFile class 117
isBTrans
in IccTerminalData class 237
isClassMemoryMgmtOn
in Functions 57
in Icc structure 57
isColor
in IccTerminalData class 238
isCommandSecurityOn
in IccTask class 203
isCommitSupported
in IccTask class 204
isCreated
in IccControl class 99
isDeletable
in IccFile class 117
isDumpAvailable
in IccAbendData class 63
isEDFOn
in Functions 57
in Icc structure 57
in IccResource class 164
isEmptyOnOpen
in IccFile class 118
isErrorSet
in IccSession class 179
isEWA
in IccTerminalData class 238
isExpired
in IccAlarmRequestId class 71
isExtended3270
in IccTerminalData class 238
in Public methods 238
isFamilySubsetEnforcementOn
in Functions 58
in Icc structure 58
isFieldOutline
in IccTerminalData class 238
in Public methods 238
isFMHContained
in IccBuf class 80
in Public methods 80
isGoodMorning
in IccTerminalData class 239
in Public methods 239
isHighlight
in IccTerminalData class 239
isKatakana
in IccTerminalData class 239
isMSRControl
in IccTerminalData class 239
isNoDataSet
in IccSession class 179
isPS
in IccTerminalData class 240
ISR2
in Example of starting transactions 22
isReadable
in IccFile class 118
in Reading ESDS records 16
in Reading KSDS records 15
in Reading RRDS records 16
isReadable method 15, 16
isRecoverable
in IccFile class 118
isResourceSecurityOn
in IccTask class 204
isRestarted
in IccTask class 204
isRouteOptionOn
in IccResource class 164
in Public methods 164
isSignalSet
in IccSession class 179
isSOSI
in IccTerminalData class 240
isStartDataAvailable
in IccTask class 204
issueAbend
in IccSession class 179
issueConfirmation
in IccSession class 179

issueError
in IccSession class 180
issuePrepare
in IccSession class 180
issueSignal
in IccSession class 180
isTextKeyboard
in IccTerminalData class 240
in Public methods 240
isTextPrint
in IccTerminalData class 240
in Public methods 240
isUpdatable
in IccFile class 119
isValidation
in IccTerminalData class 241
item (parameter)
in rewriteItem 213
in writeItem 105, 214
itemNum (parameter)
in readItem 213
in rewriteItem 213
ITMP
in Example of starting transactions 22

J

journalNum (parameter)
in Constructor 133, 137
in operator= 138
journalTypeId
in IccJournal class 133
journalTypeName (parameter)
in Constructor 139
in operator= 140
jtypeid (parameter)
in setJournalTypeId 134

K

key
complete 15
generic 15
key (parameter)
in Constructor 141
in Example of file control 18
in operator!= 143
in operator= 142
in operator== 142
keyLength
in IccFile class 119
in Reading KSDS records 15
in Writing KSDS records 16
keyLength method 15
keyPosition
in IccFile class 119
in Reading KSDS records 15
in writing KSDS records 16
keyPosition method 15
kind
in IccKey class 142
Kind
in Enumerations 144
Kind *(continued)*
in IccKey class 144
kind (parameter)
in Constructor 141
in setKind 143
KSDS
in File control 15
KSDS file 15

L

language
in IccUser class 254
language (parameter)
in setLanguage 254
lastCommand
in StateOpt 185
lastPasswordChange
in IccUser class 254
lastUseTime
in IccUser class 254
length
in IccProgram class 151
in IccRecordIndex class 158
length (parameter)
in append 78
in assign 78, 141
in Constructor 76, 77
in cut 79
in insert 80
in overlay 85
in replace 85
in setDataLength 85, 86
level (parameter)
in connectProcess 176, 177
level0
in SyncLevel 185
level1
in SyncLevel 185
level2
in SyncLevel 186
life (parameter)
in Constructor 172
lifeTime
in IccSemaphore class 173
LifeTime
in Enumerations 174
in IccSemaphore class 174
line
in Finding out information about a terminal 27
in IccTerminal class 221
lineNum (parameter)
in setLine 230
link
in IccProgram class 151
load
in IccProgram class 152
LoadOpt
in Enumerations 154
in IccProgram class 154
loc (parameter)
in Constructor 211, 212
Location
in Dynamic link library 2

Location *(continued)*
in Enumerations 215
in Header files 2
in IccTempStore class 215
in Installed contents 2
in Sample source code 2
lock
in IccSemaphore class 173
LockType
in Enumerations 174
in IccSemaphore class 174

M

main
in C++ Exceptions and the Foundation Classes 33
in Example of file control 18
in Example of managing transient data 24
in Example of polymorphic behavior 39
in Example of starting transactions 21
in Example of Temporary Storage 26
in Example of terminal control 28
in Example of time and date services 29
in Header files 2, 31
in main function 258
in Program control 19
in Storage management 40, 41
majorCode
in ConditionType 167
manual
in UpdateMode 92
Manual condition handling (noAction)
in CICS conditions 35
in Conditions, errors, and exceptions 35
maxValue
in Range 93
mem (parameter)
in initializeEnvironment 57
memory
in Location 215
message
in IccException class 111
message (parameter)
in Constructor 111
in setNextInputMessage 231
method
in Foundation Classes—reference 43
Method level
in Conditions, errors, and exceptions 37
in Platform differences 37
methodName
in IccEvent class 109
in IccException class 112
in IccMessage class 147
methodName (parameter)
in Constructor 108, 109, 111, 146
milliSeconds
in IccAbsTime class 67
in IccClock class 89
minorCode
in ConditionType 168
minutes
in IccAbsTime class 67
minutes *(continued)*
in IccTime class 242
minutes (parameter)
in Constructor 242, 244, 245, 247
in set 245, 247
Miscellaneous
Example of polymorphic behavior 39
mixed
in Case 234
mode (parameter)
in readNextRecord 129
in readPreviousRecord 129
in readRecord 120
monthOfYear
in Example of time and date services 29
in IccAbsTime class 67
in IccClock class 89
MonthOfYear
in Enumerations 92
in IccClock class 92
msg (parameter)
in clearInputMessage 151
in registerInputMessage 152
in setInputMessage 152
MVS/ESA
in ClassMemoryMgmt 59
in Storage management 41
MVSPost
in WaitPostType 211
MyTempStore
in Automatic condition handling (callHandleEvent) 36

N

N
in operatingSystem 197
name
in IccResource class 164
in IccResourceId class 168
name (parameter)
in Constructor 70, 145, 192, 216, 217, 249, 250, 256
in operator= 145, 193, 216, 218, 249, 251, 256
in setWaitText 206
nameLength
in IccResourceId class 169
NameOpt
in Enumerations 76
in IccBase class 76
netName
in IccTerminal class 221
neutral
in Color 234
new
in Storage management 40, 41
new operator 5
newPassword (parameter)
in changePassword 253
in signon 231, 232
NextTransIdOpt
in Enumerations 235
in IccTerminal class 235
noAccess
in Access 125
noAction

noAction *(continued)*
in ActionOnCondition 167
in CICS conditions 35
noCommitOnReturn
in CommitOpt 153
NONCICS
in ASRAKeyType 61
none
in FacilityType 210
noQueue
in AllocateOpt 185
normal
in ReadMode 125
in SendOpt 185
in TraceOpt 210
NoSpaceOpt
in Enumerations 215
in IccTempStore class 215
noSuspend
in Options 137
notAddable
in Access 125
NOTAPPLIC
in ASRAKeyType 61
in ASRASpaceType 62
in ASRAStorageType 63
notBrowsable
in Access 125
notDeletable
in Access 125
notPurgeable
in WaitPurgeability 211
notReadable
in Access 125
notUpdatable
in Access 125
num (parameter)
in operator!= 157
in operator« 83–85, 223, 224
in operator= 156, 170
in operator== 156
number
in IccException class 112
in IccJournalId class 138
in IccMessage class 147
in IccRBA class 157
in IccRRN class 171
in IccTask class 204
in Writing RRDS records 16
number (parameter)
in Constructor 146
in setCustomClassNum 74
numberOfItems
in IccTempStore class 212
numEvents (parameter)
in waitExternal 207
numLines (parameter)
in setNewLine 230
numRoutes (parameter)
in setRouteCodes 95

O

obj (parameter)
in Using an object 6
object
creating 5
deleting 6
in GetOpt 60
using 6
object (parameter)
in Constructor 108, 109, 111
in operator delete 74
Object level
in Conditions, errors, and exceptions 37
in Platform differences 37
objectCreationError
in C++ Exceptions and the Foundation Classes 34
in Type 113
offset (parameter)
in cut 79
in dataArea 79
in insert 80
in replace 85
in setCursor 229
onOff (parameter)
in setEDF 58, 166
open
in Status 126
openStatus
in IccFile class 119
operatingSystem
in IccSystem class 197
in Public methods 197
operatingSystemLevel
in IccSystem class 197
operator const char*
in IccBuf class 81
operator delete
in IccBase class 73
in Public methods 73
operator new
in IccBase class 74
operator!=
in IccBuf class 82
in IccKey class 143
in IccRBA class 157
in IccRRN class 171
in Public methods 82
operator«
in IccBuf class 82, 84
in IccTerminal class 221–224
in Working with IccResource subclasses 14
operator+=
in IccBuf class 81, 82
operator=
in Example of file control 18
in IccAbsTime class 67
in IccAlarmRequestId class 71
in IccBuf class 81
in IccConvId class 103
in IccDataQueueId class 107
in IccFileId class 127
in IccGroupId class 131, 132
in IccJournalId class 138
in IccJournalTypeId class 139, 140
in IccKey class 142

operator= *(continued)*
in IccLockId class 145
in IccPartnerId class 149
in IccProgramId class 154, 155
in IccRBA class 156
in IccRequestId class 160
in IccResourceId class 169
in IccRRN class 170
in IccSysId class 193
in IccTempStoreId class 216
in IccTermId class 218
in IccTimeInterval class 245
in IccTimeOfDay class 247
in IccTPNameId class 249
in IccTransId class 251
in IccUserId class 256, 257
in Protected methods 169
in Public methods 67, 245
in Working with IccResource subclasses 14
operator==
in IccBuf class 82
in IccKey class 142
in IccRBA class 156
in IccRRN class 170
opt (parameter)
in abendCode 60, 61
in access 115
in accessMethod 115
in alternateHeight 235
in alternateWidth 236
in ASRAInterrupt 61
in ASRAKeyType 61
in ASRAPSW 62
in ASRARegisters 62
in ASRASpaceType 62
in ASRAStorageType 63
in className 73
in defaultHeight 236
in defaultWidth 236
in enableStatus 116
in enterTrace 202
in graphicCharCodeSet 236
in graphicCharSetId 237
in height 220
in isAddable 117
in isAPLKeyboard 237
in isAPLText 237
in isBrowsable 117
in isBTrans 237
in isColor 238
in isDeletable 117, 118
in isDumpAvailable 63
in isEmptyOnOpen 118
in isEWA 238
in isExtended3270 238
in isFieldOutline 238
in isGoodMorning 239
in isHighlight 239
in isKatakana 239
in isMSRControl 239
in isPS 240
in isReadable 118
in isRecoverable 118
in isSOSI 240
opt (parameter) *(continued)*
in isTextKeyboard 240
in isTextPrint 240
in isUpdatable 119
in isValidation 241
in keyLength 119
in keyPosition 119
in link 151
in load 152
in openStatus 120
in originalAbendCode 64
in principalSysId 205
in priority 205
in programName 64
in recordFormat 120, 121
in recordLength 121
in rewriteItem 213
in setNextTransId 231
in type 123
in userId 207
in waitExternal 207
in width 233
in write 96
in writeAndGetReply 96
in writeItem 214
opt1 (parameter)
in abend 200
opt2 (parameter)
in abend 200
option (parameter)
in allocate 176
in retrieveData 188
in send 181, 182
in sendInvite 182
in sendLast 182, 183
in state 183
in stateText 184
in wait 135
in writeRecord 135, 136
Options
in Enumerations 136
in IccJournal class 136
options (parameter)
in Constructor 133
opts (parameter)
in setDumpOpts 205
originalAbendCode
in IccAbendData class 64
Other data sets for CICS
in Installed contents 3
overlay
in IccBuf class 85
overview of Foundation Classes 6
Overview of the foundation classes
Calling methods on a resource object 12
Creating a resource object 11

P

PA1 to PA3
in AIDVal 234
packedDecimal
in IccAbsTime class 67
Parameter level

Parameter level *(continued)*
in Conditions, errors, and exceptions 37
in Platform differences 37
parameter passing 41
Parameter passing conventions
in Miscellaneous 41
partnerName (parameter)
in Constructor 148
in operator= 149
password (parameter)
in changePassword 253
in signon 231, 232
in verifyPassword 255
passwordExpiration
in IccUser class 254
PF1 to PF24
in AIDVal 234
pink
in Color 234
PIP (parameter)
in connectProcess 176, 177
PIPList
in IccSession class 180
platform differences
method level 37
object level 37
parameter level 37
Platform differences
in Conditions, errors, and exceptions 36
Method level 37
Object level 37
Parameter level 37
platformError
in Type 113
Platforms
in Enumerations 60
in Icc structure 60
polymorphic behavior 38
Polymorphic Behavior
Example of polymorphic behavior 39
in Miscellaneous 38
popt (parameter)
in setStartOpts 190
prefix (parameter)
in registerPrefix 134
in setPrefix 134, 135
pri (parameter)
in setPriority 206
principalSysId
in IccTask class 205
in Public methods 205
print
in Polymorphic Behavior 39
priority
in IccTask class 205
in Public methods 205
process
in IccSession class 180
profile (parameter)
in Constructor 175
progName (parameter)
in Constructor 150, 154
in operator= 154, 155
program control
program control *(continued)*
example 19
introduction 19
Program control
in Using CICS Services 19
programId
in IccControl class 99
in Method level 37
in Public methods 99
programId (parameter)
in setAbendHandler 100
programName
in IccAbendData class 64
in Public methods 64
programName (parameter)
in setAbendHandler 100
Protected methods
in IccBase class 74
in IccResourceId class 169
operator= 169
setClassName 74
setCustomClassNum 74
ProtectOpt
in Enumerations 192
in IccStartRequestQ class 192
pStorage (parameter)
in freeStorage 195
Public methods
abend 200
abendCode 60
abendData 200
absTime 87
access 115
accessMethod 115
actionOnCondition 162
actionOnConditionAsChar 162
actionsOnConditionsText 163
address 150
AID 219
allocate 176
alternateHeight 235
alternateWidth 236
append 78
applName 194
ASRAInterrupt 61
ASRAKeyType 61
ASRAPSW 62
ASRARegisters 62
ASRASpaceType 62
ASRAStorageType 63
assign 78, 79, 141
beginBrowse 194, 195
beginInsert (VSAM only) 115
callingProgramId 98
cancel 186
cancelAbendHandler 98
cancelAlarm 87
changePassword 253
className 73, 109, 111, 147
classType 73, 109, 111
clear 104, 163, 212, 219
clearData 187
clearInputMessage 151
clearPrefix 133

Public methods *(continued)*
commArea 98
commitUOW 201
completeLength 141
condition 109, 163
conditionText 109, 163
connectProcess 176, 177
console 98
converse 177
convId 178
cursor 220
customClassNum 73
cut 79
data 187, 220
dataArea 79
dataAreaLength 79
dataAreaOwner 80
dataAreaType 80
dataLength 80
date 66, 88
dateFormat 195
dayOfMonth 66, 88
dayOfWeek 66, 88
daysSince1900 66, 89
daysUntilPasswordExpires 253
defaultHeight 236
defaultWidth 236
delay 201
deleteLockedRecord 116
deleteRecord 116
dump 201
empty 104, 212
enableStatus 116
endBrowse 195
endInsert (VSAM only) 117
enterTrace 202
entryPoint 151
erase 220
errorCode 178
ESMReason 253
ESMResponse 253
extractProcess 178
facilityType 202
flush 178
free 178
freeKeyboard 220
freeStorage 195, 202
get 105, 163, 179, 212, 220
getFile 196
getNextFile 196
getStorage 196, 203
graphicCharCodeSet 236
graphicCharSetId 237
groupId 253
handleEvent 163
height 220
hours 67, 242
id 164
in IccAbendData class 60
in IccAbsTime class 66
in IccAlarmRequestId class 71
in IccBase class 73
in IccBuf class 78
in IccClock class 87

Public methods *(continued)*
in IccConsole class 94
in IccControl class 98
in IccConvId class 103
in IccDataQueue class 104
in IccDataQueueId class 107
in IccEvent class 109
in IccException class 111
in IccFile class 115
in IccFileId class 127
in IccFileIterator class 129
in IccGroupId class 131
in IccJournal class 133
in IccJournalId class 138
in IccJournalTypeId class 139
in IccKey class 141
in IccLockId class 145
in IccMessage class 147
in IccPartnerId class 148
in IccProgram class 150
in IccProgramId class 154
in IccRBA class 156
in IccRecordIndex class 158
in IccRequestId class 160
in IccResource class 162
in IccResourceId class 168
in IccRRN class 170
in IccSemaphore class 173
in IccSession class 176
in IccStartRequestQ class 186
in IccSysId class 193
in IccSystem class 194
in IccTask class 200
in IccTempStore class 212
in IccTempStoreId class 216
in IccTermId class 218
in IccTerminal class 219
in IccTerminalData class 235
in IccTime class 242
in IccTimeInterval class 245
in IccTimeOfDay class 247
in IccTPNameId class 249
in IccTransId class 251
in IccUser class 253
in IccUserId class 256
initData 99
inputCursor 221
insert 80
instance 63, 94, 99, 187, 197, 203, 221
invalidPasswordAttempts 254
isAddable 117
isAPLKeyboard 237
isAPLText 237
isBrowsable 117
isBTrans 237
isColor 238
isCommandSecurityOn 203
isCommitSupported 204
isCreated 99
isDeletable 117
isDumpAvailable 63
isEDFOn 164
isEmptyOnOpen 118

Public methods *(continued)*
isErrorSet 179
isEWA 238
isExpired 71
isExtended3270 238
isFieldOutline 238
isFMHContained 80
isGoodMorning 239
isHighlight 239
isKatakana 239
isMSRControl 239
isNoDataSet 179
isPS 240
isReadable 118
isRecoverable 118
isResourceSecurityOn 204
isRestarted 204
isRouteOptionOn 164
isSignalSet 179
isSOSI 240
isStartDataAvailable 204
issueAbend 179
issueConfirmation 179
issueError 180
issuePrepare 180
issueSignal 180
isTextKeyboard 240
isTextPrint 240
isUpdatable 119
isValidation 241
journalTypeId 133
keyLength 119
keyPosition 119
kind 142
language 254
lastPasswordChange 254
lastUseTime 254
length 151, 158
lifeTime 173
line 221
link 151
load 152
lock 173
message 111
methodName 109, 112, 147
milliSeconds 67, 89
minutes 67, 242
monthOfYear 67, 89
name 164, 168
nameLength 169
netName 221
number 112, 138, 147, 157, 171, 204
numberOfItems 212
openStatus 119
operatingSystem 197
operatingSystemLevel 197
operator const char* 81
operator delete 73
operator new 74
operator!= 82, 143, 157, 171
operator« 82, 84, 221–224
operator+= 81, 82

Public methods *(continued)*
operator= 67, 71, 81, 103, 107, 127, 131, 132, 138–140, 142, 145, 149, 154–156, 160, 170, 193, 216, 218, 245, 247, 249, 251, 256, 257
operator== 82, 142, 156, 170
originalAbendCode 64
overlay 85
packedDecimal 67
passwordExpiration 254
PIPList 180
principalSysId 205
priority 205
process 180
programId 99
programName 64
put 94, 105, 134, 164, 181, 212, 224
queueName 187
readItem 105, 213
readNextItem 213
readNextRecord 129
readPreviousRecord 129
readRecord 120
receive 181, 224
receive3270Data 225
recordFormat 120
recordIndex 121
recordLength 121
registerData 187
registerInputMessage 152
registerPrefix 134
registerRecordIndex 121
release 197
releaseText 198
replace 85
replyTimeout 94
reset 129, 188
resetAbendHandler 99
resetRouteCodes 94
retrieveData 188
returnProgramId 100
returnTermId 188
returnTransId 188
rewriteItem 213
rewriteRecord 122
rollBackUOW 205
routeOption 165
run 100
seconds 68, 242
send 181, 225, 226
send3270Data 226, 227
sendInvite 182
sendLast 182, 183
sendLine 227, 228
session 100
set 245, 247
setAbendHandler 100
setAccess 122
setActionOnAnyCondition 165
setActionOnCondition 165
setActionsOnConditions 165
setAlarm 89

Public methods *(continued)*
setAllRouteCodes 95
setColor 229
setCursor 229
setData 188
setDataLength 85
setDumpOpts 205
setEDF 166
setEmptyOnOpen 122
setFMHContained 86
setHighlight 229
setInputMessage 152
setJournalTypeId 134
setKind 143
setLanguage 254
setLine 230
setNewLine 230
setNextCommArea 230
setNextInputMessage 230
setNextTransId 231
setPrefix 134
setPriority 206
setQueueName 189
setReplyTimeout 95
setReturnTermId 189
setReturnTransId 189, 190
setRouteCodes 95
setRouteOption 166
setStartOpts 190
setStatus 123
setTimerECA 71
setWaitText 206
signoff 231
signon 231, 232
start 190
startRequestQ 101
startType 206
state 183
stateText 184
summary 110, 112, 147
suspend 206
syncLevel 184
sysId 198
system 101
task 101
terminal 101
text 147
time 68, 90
timeInHours 68, 243
timeInMinutes 68, 243
timeInSeconds 68, 243
timerECA 72
transId 206
triggerDataQueueId 206
tryLock 173
type 112, 123, 158, 173, 243
typeText 112
unload 152
unlock 173
unlockRecord 123
update 90
userId 207
value 143
verifyPassword 254
Public methods *(continued)*
wait 135
waitExternal 207
waitForAID 232
waitOnAlarm 207
width 232
workArea 198, 208, 233
write 96
writeAndGetReply 96
writeItem 105, 214
writeRecord 124, 135
year 68, 90
purgeable
in WaitPurgeability 211
put
in Example of polymorphic behavior 40
in IccConsole class 94
in IccDataQueue class 105
in IccJournal class 134
in IccResource class 164
in IccSession class 181
in IccTempStore class 212
in IccTerminal class 224
in Polymorphic Behavior 39

Q

queue
in AllocateOpt 185
in NextTransIdOpt 235
queueName
in Accessing start data 21
in IccStartRequestQ class 187
queueName (parameter)
in Constructor 104, 107
in operator= 107
in setQueueName 189

R

rAbendTask
in HandleEventReturnOpt 167
Range
in Enumerations 93
in IccCondition structure 93
RBA 15
rba (parameter)
in operator!= 157
in operator= 156
in operator== 156
rContinue
in HandleEventReturnOpt 167
readable
in Access 125
reading data 24
Reading data
in Transient Data 24
in Using CICS Services 24
Reading ESDS records
in File control 16
in Reading records 16
reading items 25
Reading items

Reading items *(continued)*
in Temporary storage 25
in Using CICS Services 25
Reading KSDS records
in File control 15
in Reading records 15
Reading records
in File control 15
in Using CICS Services 15
Reading ESDS records 16
Reading KSDS records 15
Reading RRDS records 16
Reading RRDS records
in File control 16
in Reading records 16
readItem
in Example of Temporary Storage 26
in IccDataQueue class 105
in IccTempStore class 213
in Reading data 24
in Reading items 25
in Scope of data in IccBuf reference returned from 'read' methods 42
in Temporary storage 25
in Transient Data 23
in Working with IccResource subclasses 14
ReadMode
in Enumerations 125
in IccFile class 125
readNextItem
in IccTempStore class 213
in Scope of data in IccBuf reference returned from 'read' methods 42
in Temporary storage 25
readNextRecord
in Browsing records 17
in IccFileIterator class 129
in Public methods 129
readNextRecord method 17
READONLY
in ASRAStorageType 63
readPreviousRecord
in Browsing records 17
in IccFileIterator class 129
readRecord
in C++ Exceptions and the Foundation Classes 34
in Deleting locked records 17
in IccFile class 120
in Reading records 15
in Updating records 17
readRecord method 15
receive
in IccSession class 181
in IccTerminal class 224
in Receiving data from a terminal 27
receive3270data
in Receiving data from a terminal 27
receive3270Data
in IccTerminal class 225
in Public methods 225
receiving data from a terminal 27
Receiving data from a terminal
in Terminal control 27
in Using CICS Services 27
record (parameter)
in writeRecord 135
recordFormat
in IccFile class 120
in Reading ESDS records 16
in Reading RRDS records 16
in Writing ESDS records 16
in Writing RRDS records 17
recordFormat method 16
recordIndex
in IccFile class 121
in Reading ESDS records 16
in Reading KSDS records 15
in Reading RRDS records 16
in Writing ESDS records 16
in Writing KSDS records 16
in Writing RRDS records 17
recordIndex method 15, 16
recordLength
in IccFile class 121
in Reading ESDS records 16
in Reading KSDS records 15
in Reading RRDS records 16
in Writing ESDS records 16
in Writing KSDS records 16
in Writing RRDS records 17
recordLength method 15, 16
red
in Color 234
registerData
in Example of starting transactions 22
in IccStartRequestQ class 187
in Starting transactions 20
registerInputMessage
in IccTerminal class 152
registerPrefix
in IccJournal class 134
in Public methods 134
registerRecordIndex
in IccFile class 121
in Reading ESDS records 16
in Reading KSDS records 15
in Reading RRDS records 16
in Writing ESDS records 16
in Writing KSDS records 16
in Writing records 16
in Writing RRDS records 17
registerRecordIndex method 15
relative byte address 15
relative record number 15
release
in IccSystem class 197
releaseAtTaskEnd
in LoadOpt 154
releaseText
in IccSystem class 198
remoteTermId
in Example of starting transactions 22
replace
in IccBuf class 85
in IccBuf constructors 13
replyTimeout
in IccConsole class 94
req

req *(continued)*
in Example of starting transactions 22
req1
in Example of starting transactions 21
req2
in Example of starting transactions 22
reqestName (parameter)
in operator= 160
reqId (parameter)
in cancel 186
in cancelAlarm 88
in delay 201
in setAlarm 89
in start 190, 191
requestName (parameter)
in Constructor 160
in operator= 71, 160
requestNum (parameter)
in wait 135
reset
in Browsing records 17
in IccFileIterator class 129
in IccStartRequestQ class 188
resetAbendHandler
in IccControl class 99
resetRouteCodes
in IccConsole class 94
in Public methods 94
resId (parameter)
in beginBrowse 194, 195
resName (parameter)
in beginBrowse 195
in Constructor 168
resource (parameter)
in beginBrowse 194, 195
in Constructor 172
in endBrowse 195
in enterTrace 202
resource class 8
Resource classes
in Overview of the foundation classes 8
resource identification class 7
Resource identification classes
in Overview of the foundation classes 7
resource object
creating 11
ResourceType
in Enumerations 199
in IccSystem class 199
respectAbendHandler
in AbendHandlerOpt 209
retrieveData
in Accessing start data 21
in IccStartRequestQ class 186, 188
in Mapping EXEC CICS calls to Foundation Class methods 44
RetrieveOpt
in Enumerations 192
in IccStartRequestQ class 192
return
in Mapping EXEC CICS calls to Foundation Class methods 44
returnCondition
in NoSpaceOpt 215
returnProgramId
in IccControl class 100
in Public methods 100
returnTermId
in Accessing start data 21
in IccStartRequestQ class 188
returnToCICS
in Functions 58
in Icc structure 58
returnTransId
in Accessing start data 21
in IccStartRequestQ class 188
reverse
in Highlight 234
rewriteItem
in Example of Temporary Storage 26
in IccTempStore class 213
in Temporary storage 25
in Updating items 25
in Writing items 25
rewriteRecord
in IccFile class 122
in Updating records 17
rewriteRecord method 17
rewriting records 17
rollBackUOW
in IccTask class 205
routeOption
in IccResource class 165
row (parameter)
in send 226
in setCursor 229
RRDS file
in File control 15
RRN 15
rrn (parameter)
in operator!= 171
in operator= 170
in operator== 170
rThrowException
in HandleEventReturnOpt 167
run
in Base classes 7
in C++ Exceptions and the Foundation Classes 33
in Example of file control 18, 19
in Example of managing transient data 24, 25
in Example of polymorphic behavior 39
in Example of starting transactions 21
in Example of Temporary Storage 26, 27
in Example of terminal control 28
in Example of time and date services 29
in IccControl class 97, 100
in main function 258
in Mapping EXEC CICS calls to Foundation Class methods 44
in Program control 19
Running the sample applications 3

S

sample source 2
Sample source code
in Installed contents 2
Location 2

scope of data 41
Scope of data in IccBuf reference returned from 'read' methods
in Miscellaneous 41
scope of references 41
SDFHLOAD 3
SDFHPROC 3
SDFHSDCK 3
search (parameter)
in Constructor 128
in reset 130
SearchCriterion
in Enumerations 126
in IccFile class 126
seconds
in IccAbsTime class 68
in IccTime class 242
seconds (parameter)
in Constructor 242, 244, 245, 247
in set 245, 247, 248
in setReplyTimeout 95
send
in Example of terminal control 28
in IccSession class 181
in IccTerminal class 225, 226
send (parameter)
in converse 177
in put 94
in send 181
in sendInvite 182
in sendLast 182
in write 96
in writeAndGetReply 96
send3270Data
in IccTerminal class 226, 227
sending data to a terminal 27
Sending data to a terminal
in Terminal control 27
in Using CICS Services 27
sendInvite
in IccSession class 182
sendLast
in IccSession class 182, 183
sendLine
in Example of file control 18
in Example of terminal control 28
in IccTerminal class 227, 228
SendOpt
in Enumerations 185
in IccSession class 185
sequential reading of files 17
session
in FacilityType 210
in IccControl class 100
set
in IccTimeInterval class 245
in IccTimeOfDay class 247
set (parameter)
in boolText 56
set...
in Sending data to a terminal 27
setAbendHandler
in IccControl class 100
setAccess
setAccess *(continued)*
in IccFile class 122
setActionOnAnyCondition
in IccResource class 165
setActionOnCondition
in IccResource class 165
setActionsOnConditions
in IccResource class 165
setAlarm
in IccAlarmRequestId class 70
in IccClock class 89
setAllRouteCodes
in IccConsole class 95
setClassName
in IccBase class 74
in Protected methods 74
setColor
in Example of terminal control 28
in IccTerminal class 229
setCursor
in IccTerminal class 229
setCustomClassNum
in IccBase class 74
in Protected methods 74
setData
in IccStartRequestQ class 188
in Starting transactions 20
setDataLength
in IccBuf class 85
setDumpOpts
in IccTask class 205
setEDF
in Functions 58
in Icc structure 58
in IccResource class 166
setEmptyOnOpen
in IccFile class 122
in Public methods 122
setFMHContained
in IccBuf class 86
in Public methods 86
setHighlight
in Example of terminal control 28
in IccTerminal class 229
setInputMessage
in IccProgram class 152
in Public methods 152
setJournalTypeId
in IccJournal class 134
setKind
in Example of file control 18
in IccKey class 143
setLanguage
in IccUser class 254
setLine
in IccTerminal class 230
setNewLine
in IccTerminal class 230
setNextCommArea
in IccTerminal class 230
in Public methods 230
setNextInputMessage
in IccTerminal class 230
setNextTransId

setNextTransId *(continued)*
in IccTerminal class 231
setPrefix
in IccJournal class 134
setPriority
in IccTask class 206
in Public methods 206
setQueueName
in Example of starting transactions 22
in IccStartRequestQ class 189
in Starting transactions 20
setReplyTimeout
in IccConsole class 95
setReturnTermId
in Example of starting transactions 22
in IccStartRequestQ class 189
in Starting transactions 20
setReturnTransId
in Example of starting transactions 22
in IccStartRequestQ class 189, 190
in Starting transactions 21
setRouteCodes
in IccConsole class 95
setRouteOption
in Example of starting transactions 22, 23
in IccResource class 166
in Program control 20
in Public methods 166
setStartOpts
in IccStartRequestQ class 190
setStatus
in IccFile class 123
setTimerECA
in IccAlarmRequestId class 71
setWaitText
in IccTask class 206
Severe error handling (abendTask)
in CICS conditions 36
in Conditions, errors, and exceptions 36
SeverityOpt
in Enumerations 97
in IccConsole class 97
signoff
in IccTerminal class 231
signon
in IccTerminal class 231, 232
in Public methods 231, 232
singleton class 11
Singleton classes
in Creating a resource object 11
in Using CICS resources 11
size (parameter)
in getStorage 196, 197, 203
in operator new 74
start
in Example of starting transactions 23
in IccRequestId class 159
in IccStartRequestQ class 186, 190
in Mapping EXEC CICS calls to Foundation Class methods 44
in Parameter passing conventions 41
in Starting transactions 21
Starting transactions
in Starting transactions asynchronously 20
Starting transactions *(continued)*
in Using CICS Services 20
starting transactions asynchronously 20
Starting transactions asynchronously
Accessing start data 21
Cancelling unexpired start requests 21
Example of starting transactions 21
in Using CICS Services 20
Starting transactions 20
startIO
in Options 137
startRequest
in StartType 210
startRequestQ
in Example of starting transactions 22, 23
in IccControl class 101
startType
in Example of starting transactions 23
in IccTask class 206
StartType
in Enumerations 210
in IccTask class 210
state
in IccSession class 183
StateOpt
in Enumerations 185
in IccSession class 185
stateText
in IccSession class 184
Status
in Enumerations 126
in IccFile class 126
status (parameter)
in setStatus 123
Storage management
in Miscellaneous 40
StorageOpts
in Enumerations 210
in IccTask class 210
storageOpts (parameter)
in getStorage 196, 197, 203
storeName (parameter)
in Constructor 211
SUBSPACE
in ASRASpaceType 62
summary
in IccEvent class 110
in IccException class 112
in IccMessage class 147
support classes 10
Support Classes
in Overview of the foundation classes 10
suppressDump
in AbendDumpOpt 209
suspend
in IccTask class 206
in NoSpaceOpt 215
synchronous
in Options 137
syncLevel
in IccSession class 184
SyncLevel
in Enumerations 185
in IccSession class 185

sysId
in IccSystem class 198
sysId (parameter)
in Constructor 175
in setRouteOption 166
sysName (parameter)
in Constructor 175
in setRouteOption 166
system
in IccControl class 101

T

task
in IccControl class 101
in LifeTime 174
temporary storage
deleting items 26
example 26
introduction 25
reading items 25
updating items 25
Writing items 25
Temporary storage
Deleting items 26
Example of Temporary Storage 26
in Using CICS Services 25
Reading items 25
Updating items 25
Writing items 25
termId (parameter)
in setReturnTermId 189
in start 190, 191
terminal
finding out about 27
in FacilityType 210
in IccControl class 101
receiving data from 27
sending data to 27
terminal control
example 27
finding out information 27
introduction 27
receiving data 27
sending data 27
Terminal control
Example of terminal control 27
Finding out information about a terminal 27
in Using CICS Services 27
Receiving data from a terminal 27
Sending data to a terminal 27
terminalInput
in StartType 210
termName (parameter)
in setReturnTermId 189
Test
in C++ Exceptions and the Foundation Classes 33
test (parameter)
in boolText 56
text
in IccMessage class 147
text (parameter)
in Constructor 77, 146
in operator!= 143
text (parameter) *(continued)*
in operator« 83, 222, 223
in operator+= 82
in operator= 81
in operator== 142, 143
in writeItem 106, 214
throw
in C++ Exceptions and the Foundation Classes 32
in Exception handling (throwException) 36
throwException
in ActionOnCondition 167
in CICS conditions 35
ti
in Example of starting transactions 22, 23
time
in IccAbsTime class 68
in IccClock class 90
time (parameter)
in Constructor 65, 245, 247
in delay 201
in setAlarm 89
in start 190, 191
Time and date services
Example of time and date services 29
in Using CICS Services 28
time services 28
timeInHours
in IccAbsTime class 68
in IccTime class 243
timeInMinutes
in IccAbsTime class 68
in IccTime class 243
timeInSeconds
in IccAbsTime class 68
in IccTime class 243
timeInterval
in Type 244
timeInterval (parameter)
in operator= 245
timeOfDay
in Type 244
timeOfDay (parameter)
in operator= 247
timerECA
in IccAlarmRequestId class 72
timerECA (parameter)
in Constructor 70
in setTimerECA 71
timeSeparator (parameter)
in time 68, 90
TPName (parameter)
in connectProcess 177
traceNum (parameter)
in enterTrace 202
TraceOpt
in Enumerations 210
in IccTask class 210
tracing
activating trace output 32
transId
in IccTask class 206
transid (parameter)
in setNextTransId 231
transId (parameter)

transId (parameter) *(continued)*
in cancel 186
in connectProcess 177
in link 151
in setNextTransId 231
in setReturnTransId 189
in start 190, 191
transient data
deleting queues 24
example 24
introduction 23
reading data 24
Writing data 24
Transient Data
Deleting queues 24
Example of managing transient data 24
in Using CICS Services 23
Reading data 24
Writing data 24
transName (parameter)
in setReturnTransId 190
triggerDataQueueId
in IccTask class 206
trueFalse (parameter)
in setEmptyOnOpen 122
try
in C++ Exceptions and the Foundation Classes 32, 33
in Exception handling (throwException) 36
in main function 258
tryLock
in IccSemaphore class 173
tryNumber
in C++ Exceptions and the Foundation Classes 33
type
in C++ Exceptions and the Foundation Classes 34
in IccException class 112
in IccFile class 123
in IccRecordIndex class 158
in IccSemaphore class 173
in IccTime class 243
Type
in Enumerations 113, 159, 244
in IccException class 113
in IccRecordIndex class 159
in IccTime class 244
type (parameter)
in condition 109, 163
in Constructor 72, 73, 76, 77, 158, 168, 172
in waitExternal 207
typeText
in IccException class 112

U

underscore
in Highlight 234
UNIX
in ClassMemoryMgmt 59
in Storage management 41
unknownException
in Functions 58
in Icc structure 58
unload
in IccProgram class 152
unlock
in IccSemaphore class 173
unlockRecord
in IccFile class 123
UOW
in LifeTime 174
updatable
in Access 125
update
in IccClock class 90
in ReadMode 126
update (parameter)
in Constructor 87
UpdateMode
in Enumerations 92
in IccClock class 92
updateToken (parameter)
in deleteLockedRecord 116
in readNextRecord 129
in readPreviousRecord 129
in readRecord 120
in rewriteRecord 122
in unlockRecord 123, 124
updating items 25
Updating items
in Temporary storage 25
in Using CICS Services 25
updating records 17
Updating records
in File control 17
in Using CICS Services 17
upper
in Case 234
USER
in ASRAStorageType 63
user (parameter)
in signon 232
userDataKey
in StorageOpts 210
USEREXECKEY
in ASRAKeyType 61
userId
in IccTask class 207
userId (parameter)
in start 191
userName (parameter)
in Constructor 252
Using an object
in C++ Objects 6
using CICS resources 11
Using CICS resources
Calling methods on a resource object 12
Creating a resource object 11
in Overview of the foundation classes 11
Singleton classes 11
Using CICS Services
Accessing start data 21
Browsing records 17
Cancelling unexpired start requests 21
Deleting items 26
Deleting queues 24
Deleting records 17
Example of file control 17
Example of managing transient data 24

Using CICS Services *(continued)*
Example of starting transactions 21
Example of Temporary Storage 26
Example of terminal control 27
Example of time and date services 29
Finding out information about a terminal 27
Reading data 24
Reading items 25
Reading records 15
Receiving data from a terminal 27
Sending data to a terminal 27
Starting transactions 20
Updating items 25
Updating records 17
Writing data 24
Writing items 25
Writing records 16

V

value
in IccKey class 143
value (parameter)
in operator= 142
variable (parameter)
in Foundation Classes—reference 43
verifyPassword
in IccUser class 254
in Public methods 254
VSAM 15

W

wait
in IccJournal class 135
in SendOpt 185
waitExternal
ECBList (parameter)
in waitExternal 207
in IccTask class 207
numEvents (parameter)
in waitExternal 207
opt (parameter)
in waitExternal 207
type (parameter)
in waitExternal 207
waitForAID
in Example of terminal control 28
in IccTerminal class 232
waitOnAlarm
in IccAlarmRequestId class 70
in IccTask class 207
WaitPostType
in Enumerations 211
in IccTask class 211
WaitPurgeability
in Enumerations 211
in IccTask class 211
width
in IccTerminal class 232
workArea
in IccSystem class 198
workArea *(continued)*
in IccTask class 208
in IccTerminal class 233
Working with IccResource subclasses
in Buffer objects 14
in IccBuf class 14
write
in IccConsole class 96
writeAndGetReply
in IccConsole class 96
writeItem
in C++ Exceptions and the Foundation Classes 34
in Calling methods on a resource object 12
in IccDataQueue class 105
in IccTempStore class 214
in Temporary storage 25
in Transient Data 23
in Working with IccResource subclasses 14
in Writing data 24
in Writing items 25
writeRecord
in Example of file control 18
in IccFile class 124
in IccJournal class 135
in Writing KSDS records 16
in Writing records 16
in Writing RRDS records 16
writeRecord method
IccFile class 16
Writing data
in Transient Data 24
in Using CICS Services 24
Writing ESDS records
in File control 16
in Writing records 16
Writing items
in Temporary storage 25
in Using CICS Services 25
Writing KSDS records
in File control 16
in Writing records 16
Writing records
in File control 16
in Using CICS Services 16
Writing ESDS records 16
Writing KSDS records 16
Writing RRDS records 16
Writing RRDS records
in File control 16
in Writing records 16

X

X
in actionOnConditionAsChar 162
in operatingSystem 197
XPLINK 3

Y

year
in IccAbsTime class 68
in IccClock class 90

yellow
in Color 234
yesNo (parameter)
in setFMHContained 86