CICS® Transaction Server for z/OS™

C++ OO Class Libraries
## Contents

Preface ................................................................. xxv
Who this book is for .............................................. xxv
What this book is about ......................................... xxv
What you need to know before reading this book .......... xxvi
Notes on terminology .............................................. xxvi

Part 1. Installation and setup ........................................ 1

Chapter 1. Getting ready for object oriented CICS .............. 3

Chapter 2. Installed contents ....................................... 5
Header files ......................................................... 5
Location ............................................................... 6
Dynamic link library ............................................... 6
Location ............................................................... 6
Sample source code ................................................ 6
Location ............................................................... 6
Running the sample applications ................................ 6
Other datasets for CICS Transaction Server for z/OS ........... 6

Chapter 3. Hello World ............................................... 9
Compile and link "Hello World" ................................... 10
Running "Hello World" on your CICS server .................... 10
Expected Output from "Hello World" ............................ 10

Part 2. Using the CICS foundation classes ...................... 11

Chapter 4. C++ Objects .............................................. 13
Creating an object .................................................. 13
Using an object ...................................................... 14
Deleting an object ................................................... 14

Chapter 5. Overview of the foundation classes .................. 15
Base classes .......................................................... 15
Resource identification classes .................................. 16
Resource classes ...................................................... 17
Support Classes ....................................................... 18
Using CICS resources .............................................. 19
Creating a resource object ....................................... 19
Calling methods on a resource object ......................... 20

Chapter 6. Buffer objects .......................................... 23
IccBuf class .......................................................... 23
Data area ownership ............................................... 23
Data area extensibility ............................................. 23
IccBuf constructors ................................................ 23
IccBuf methods ....................................................... 24
Working with IccResource subclasses ......................... 24

Chapter 7. Using CICS Services .................................... 27
File control ............................................................ 27
Reading records ...................................................... 27
## Part 3. Foundation Classes—reference

### Chapter 11. Icc structure

- Functions
  - boolText
  - catchException
  - conditionText
  - initializeEnvironment
  - isClassMemoryMgmtOn
  - isEDFOn
  - isFamilySubsetEnforcementOn
  - returnToCICS
  - setEDF
  - unknownException

- Enumerations
  - Bool
  - BoolSet
  - ClassMemoryMgmt
  - FamilySubset
  - GetOpt
  - Platforms

### Chapter 12. IccAbendData class

- IccAbendData constructor (protected)
  - Constructor

- Public methods
  - abendCode
  - ASRAInterrupt
  - ASRAKeyType
  - ASRAPSW
  - ASRAResisters
  - ASRASpaceType
  - ASRASorageType
  - instance
  - isDumpAvailable
  - originalAbendCode
  - programName

- Inherited public methods

- Inherited protected methods

### Chapter 13. IccAbsTime class

- IccAbsTime constructor
  - Constructor (1)
  - Constructor (2)

- Public methods
  - date
  - dayOfMonth
  - dayOfWeek
  - daysSince1900
  - hours
  - milliSeconds
  - minutes
  - monthOfYear

- operator="
Chapter 14. IccAlarmRequestId class

IccAlarmRequestId constructors

Constructor (1) 
Constructor (2) 
Constructor (3) 

Public methods

isExpired
operator= (1)
operator= (2)
operator= (3)
setTimerECA
timerECA

Inherited public methods

Inherited protected methods

Chapter 15. IccBase class

IccBase constructor (protected) 

Constructor 

Public methods

classType
className
customClassNum
operator delete
operator new

Protected methods

setClassName
setCustomClassNum

Enumerations

ClassType
NameOpt

Chapter 16. IccBuf class

IccBuf constructors

Constructor (1)
Constructor (2)
Constructor (3)
Constructor (4)

Public methods

append (1)
append (2)
assign (1)
assign (2)
cut
dataArea
dataAreaLength
dataAreaOwner
dataAreaType ........................................ 85
dataLength ........................................... 85
insert .................................................. 86
isFMHContained ....................................... 86
operator const char* ................................. 86
operator= (1) ......................................... 86
operator= (2) ......................................... 86
operator+= (1) ......................................... 87
operator+= (2) ......................................... 87
operator== ............................................ 87
operator!= ............................................. 87
operator<< (1) ......................................... 87
operator<< (2) ......................................... 87
operator<< (3) ......................................... 87
operator<< (4) ......................................... 88
operator<< (5) ......................................... 88
operator<< (6) ......................................... 88
operator<< (7) ......................................... 88
operator<< (8) ......................................... 88
operator<< (9) ......................................... 88
operator<< (10) ........................................ 88
operator<< (11) ........................................ 88
operator<< (12) ........................................ 88
operator<< (13) ........................................ 88
operator<< (14) ........................................ 88
operator<< (15) ........................................ 88
overlay ............................................... 89
replace ............................................... 89
setDataLength ......................................... 89
setFMHContained ...................................... 89
Inherited public methods ............................ 89
Inherited protected methods ......................... 90
Enumerations .......................................... 90
DataAreaOwner ........................................ 90
DataAreaType .......................................... 90

Chapter 17. IccClock class .......................... 91
IccClock constructor .................................. 91
Constructor ........................................... 91
Public methods ........................................ 91
absTime .............................................. 91
cancelAlarm .......................................... 91
date .................................................. 91
dayOfMonth ............................................ 92
dayOfWeek ............................................ 92
daysSince1900 ........................................ 92
milliSeconds ......................................... 92
monthOfYear .......................................... 92
setAlarm .............................................. 93
time .................................................. 93
update ............................................... 93
year .................................................... 93
Inherited public methods ............................ 93
Inherited protected methods ......................... 94
Enumerations .......................................... 94
DateFormat ........................................... 94
<table>
<thead>
<tr>
<th>Chapter 18. IccCondition structure</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerations</td>
<td>97</td>
</tr>
<tr>
<td>Codes</td>
<td>97</td>
</tr>
<tr>
<td>Range</td>
<td>97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 19. IccConsole class</th>
<th>99</th>
</tr>
</thead>
<tbody>
<tr>
<td>IccConsole constructor (protected)</td>
<td>99</td>
</tr>
<tr>
<td>Constructor</td>
<td>99</td>
</tr>
<tr>
<td>Public methods</td>
<td>99</td>
</tr>
<tr>
<td>instance</td>
<td>99</td>
</tr>
<tr>
<td>put</td>
<td>99</td>
</tr>
<tr>
<td>replyTimeout</td>
<td>99</td>
</tr>
<tr>
<td>resetRouteCodes</td>
<td>100</td>
</tr>
<tr>
<td>setAllRouteCodes</td>
<td>100</td>
</tr>
<tr>
<td>setReplyTimeout (1)</td>
<td>100</td>
</tr>
<tr>
<td>setReplyTimeout (2)</td>
<td>100</td>
</tr>
<tr>
<td>setRouteCodes</td>
<td>100</td>
</tr>
<tr>
<td>write</td>
<td>100</td>
</tr>
<tr>
<td>writeAndGetReply</td>
<td>101</td>
</tr>
<tr>
<td>Inherited public methods</td>
<td>101</td>
</tr>
<tr>
<td>Inherited protected methods</td>
<td>101</td>
</tr>
<tr>
<td>Enumerations</td>
<td>102</td>
</tr>
<tr>
<td>SeverityOpt</td>
<td>102</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 20. IccControl class</th>
<th>103</th>
</tr>
</thead>
<tbody>
<tr>
<td>IccControl constructor (protected)</td>
<td>103</td>
</tr>
<tr>
<td>Constructor</td>
<td>103</td>
</tr>
<tr>
<td>Public methods</td>
<td>103</td>
</tr>
<tr>
<td>callingProgramId</td>
<td>103</td>
</tr>
<tr>
<td>cancelAbendHandler</td>
<td>103</td>
</tr>
<tr>
<td>commArea</td>
<td>103</td>
</tr>
<tr>
<td>console</td>
<td>104</td>
</tr>
<tr>
<td>initData</td>
<td>104</td>
</tr>
<tr>
<td>instance</td>
<td>104</td>
</tr>
<tr>
<td>isCreated</td>
<td>104</td>
</tr>
<tr>
<td>programId</td>
<td>104</td>
</tr>
<tr>
<td>resetAbendHandler</td>
<td>104</td>
</tr>
<tr>
<td>returnProgramId</td>
<td>105</td>
</tr>
<tr>
<td>run</td>
<td>105</td>
</tr>
<tr>
<td>session</td>
<td>105</td>
</tr>
<tr>
<td>setAbendHandler (1)</td>
<td>105</td>
</tr>
<tr>
<td>setAbendHandler (2)</td>
<td>105</td>
</tr>
<tr>
<td>startRequestQ</td>
<td>105</td>
</tr>
<tr>
<td>system</td>
<td>105</td>
</tr>
<tr>
<td>task</td>
<td>106</td>
</tr>
<tr>
<td>terminal</td>
<td>106</td>
</tr>
<tr>
<td>Inherited public methods</td>
<td>106</td>
</tr>
<tr>
<td>Inherited protected methods</td>
<td>106</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 21. IccConvId class</th>
<th>107</th>
</tr>
</thead>
<tbody>
<tr>
<td>IccConvId constructors</td>
<td>107</td>
</tr>
<tr>
<td>Constructor (1)</td>
<td>107</td>
</tr>
<tr>
<td>Chapter 22. IccDataQueue class</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--</td>
</tr>
<tr>
<td>IccDataQueue constructors</td>
<td>109</td>
</tr>
<tr>
<td>Constructor (1)</td>
<td>109</td>
</tr>
<tr>
<td>Constructor (2)</td>
<td>109</td>
</tr>
<tr>
<td>Public methods</td>
<td>109</td>
</tr>
<tr>
<td>clear</td>
<td>109</td>
</tr>
<tr>
<td>empty</td>
<td>109</td>
</tr>
<tr>
<td>get</td>
<td>109</td>
</tr>
<tr>
<td>put</td>
<td>110</td>
</tr>
<tr>
<td>readItem</td>
<td>110</td>
</tr>
<tr>
<td>writeItem (1)</td>
<td>110</td>
</tr>
<tr>
<td>writeItem (2)</td>
<td>110</td>
</tr>
<tr>
<td>Inherited public methods</td>
<td>110</td>
</tr>
<tr>
<td>Inherited protected methods</td>
<td>111</td>
</tr>
</tbody>
</table>

| Chapter 23. IccDataQueueId class |  |
|-----------------------------------|---|---|
| IccDataQueueId constructors     | 113 |  |
| Constructor (1)                 | 113 |  |
| Constructor (2)                 | 113 |  |
| Public methods                   | 113 |  |
| operator= (1)                    | 113 |  |
| operator= (2)                    | 113 |  |
| Inherited public methods         | 113 |  |
| Inherited protected methods      | 114 |  |

| Chapter 24. IccEvent class     |  |
|---------------------------------|---|---|
| IccEvent constructor            | 115 |  |
| Constructor                     | 115 |  |
| Public methods                   | 115 |  |
| className                       | 115 |  |
| classType                       | 115 |  |
| condition                       | 115 |  |
| conditionText                   | 116 |  |
| methodName                      | 116 |  |
| summary                         | 116 |  |
| Inherited public methods         | 116 |  |
| Inherited protected methods      | 116 |  |

| Chapter 25. IccException class |  |
|---------------------------------|---|---|
| IccException constructor       | 117 |  |
| Constructor                     | 117 |  |
| Public methods                   | 117 |  |
| className                       | 118 |  |
| classType                       | 118 |  |
| message                         | 118 |  |
| methodName                      | 118 |  |
| number                          | 118 |  |
| summary                         | 118 |  |
| type                            | 118 |  |
Inherited protected methods ........................................... 132

Chapter 28. IccFileIterator class ..................................... 133
IccFileIterator constructor ............................................ 133
  Constructor ......................................................... 133
Public methods ......................................................... 133
  readNextRecord ................................................... 133
  readPreviousRecord ............................................... 134
  reset .............................................................. 134
Inherited public methods ............................................ 134
Inherited protected methods ....................................... 135

Chapter 29. IccGroupId class ........................................... 137
IccGroupId constructors .............................................. 137
  Constructor (1) .................................................... 137
  Constructor (2) .................................................... 137
Public methods ......................................................... 137
  operator= (1) ....................................................... 137
  operator= (2) ....................................................... 137
Inherited public methods ............................................ 137
Inherited protected methods ....................................... 138

Chapter 30. IccJournal class .......................................... 139
IccJournal constructors .............................................. 139
  Constructor (1) .................................................... 139
  Constructor (2) .................................................... 139
Public methods ......................................................... 139
  clearPrefix ....................................................... 139
  journalTypeld ................................................... 140
  put ............................................................... 140
  registerPrefix ................................................... 140
  setJournalTypeld (1) .............................................. 140
  setJournalTypeld (2) .............................................. 140
  setPrefix (1) ...................................................... 140
  setPrefix (2) ...................................................... 140
  wait ............................................................... 140
  writeRecord (1) ................................................... 141
  writeRecord (2) ................................................... 141
Inherited public methods ............................................ 141
Inherited protected methods ....................................... 142
Enumerations .......................................................... 142
  Options ........................................................... 142

Chapter 31. IccJournalId class ........................................ 143
IccJournalId constructors ............................................ 143
  Constructor (1) .................................................... 143
  Constructor (2) .................................................... 143
Public methods ......................................................... 143
  number ............................................................ 143
  operator= (1) ....................................................... 143
  operator= (2) ....................................................... 143
Inherited public methods ............................................ 144
Inherited protected methods ....................................... 144

Chapter 32. IccJournalTypeld class ................................ 145
IccJournalTypeld constructors ...................................... 145
  Constructor ....................................................... 145
Chapter 33. IccKey class

IccKey constructors

Constructor (1) ................................ 147
Constructor (2) ................................ 147
Constructor (3) ................................ 147

Public methods

assign ........................................ 147
completeLength ............................ 147
kind ......................................... 147
operator= (1) ............................... 147
operator= (2) ............................... 147
operator= (3) ............................... 147
operator== (1) .............................. 147
operator== (2) .............................. 147
operator== (3) .............................. 147
operator!= (1) .............................. 147
operator!= (2) .............................. 147
operator!= (3) .............................. 147
setKind ..................................... 147
value ........................................ 147

Inherited public methods

Inherited protected methods

Enumerations

Kind ........................................ 149

Chapter 34. IccLockId class

IccLockId constructors

Constructor (1) ................................ 151
Constructor (2) ................................ 151

Public methods

operator= (1) ............................... 151
operator= (2) ............................... 151
operator!= (1) .............................. 151
operator!= (2) .............................. 151

Inherited public methods

Inherited protected methods

Chapter 35. IccMessage class

IccMessage constructor

Constructor .................................. 153

Public methods

className .................................. 153
methodName ................................ 153
number ..................................... 153
summary .................................... 153
text ........................................ 154

Inherited public methods

Inherited protected methods

Chapter 36. IccPartnerId class .................................. 155
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Class</th>
<th>Sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>IccProgram class</td>
<td>IccProgram constructors, Public methods, Inherited public methods, Inherited protected methods, Enumerations</td>
</tr>
<tr>
<td>38</td>
<td>IccProgramId class</td>
<td>IccProgramId constructors, Public methods, Inherited public methods, Inherited protected methods, Enumerations</td>
</tr>
<tr>
<td>39</td>
<td>IccRBA class</td>
<td>IccRBA constructor, Public methods, Inherited public methods, Inherited protected methods</td>
</tr>
<tr>
<td>40</td>
<td>IccRecordIndex class</td>
<td>IccRecordIndex constructor (protected), Public methods, Inherited public methods, Inherited protected methods</td>
</tr>
</tbody>
</table>
Public methods ........................................... 165
  length ........................................... 165
  type ........................................... 165
Inherited public methods .............................. 165
Inherited protected methods ............................ 166
Enumerations ........................................... 166
  Type ........................................... 166

Chapter 41. IccRequestId class ............................ 167
IccRequestId constructors .................................. 167
  Constructor (1) .................................. 167
  Constructor (2) .................................. 167
  Constructor (3) .................................. 167
Public methods ........................................... 167
  operator= (1) .................................. 167
  operator= (2) .................................. 167
Inherited public methods .................................. 168
Inherited protected methods ............................. 168

Chapter 42. IccResource class ............................ 169
IccResource constructor (protected) ................. 169
  Constructor ....................................... 169
Public methods ........................................... 169
  actionOnCondition .................................. 169
  actionOnConditionAsChar ............................. 169
  actionsOnConditionsText ............................ 170
  clear ........................................... 170
  condition ........................................ 170
  conditionText ...................................... 170
  get ............................................... 170
  handleEvent ....................................... 171
  id ................................................ 171
  isEDFOn .......................................... 171
  isRouteOptionOn .................................... 171
  name ............................................. 171
  put ............................................... 171
  routeOption ....................................... 172
  setActionOnAnyCondition ......................... 172
  setActionOnCondition .................................. 172
  setActionsOnConditions ......................... 172
  setEDF ........................................... 172
  setRouteOption (1) .................................. 173
  setRouteOption (2) .................................. 173
Inherited public methods .................................... 173
Inherited protected methods ............................. 173
Enumerations ........................................... 173
  ActionOnCondition .................................. 173
  HandleEventReturnOpt ............................. 174
  ConditionType ...................................... 174

Chapter 43. IccResourceld class ............................ 175
IccResourceld constructors (protected) ........... 175
  Constructor (1) .................................. 175
  Constructor (2) .................................. 175
Public methods ........................................... 175
  name ............................................. 175
isNoDataSet ........................................... 186
isSignalSet ............................................ 186
issueAbend ............................................. 186
issueConfirmation ..................................... 187
issueError .............................................. 187
issuePrepare ........................................... 187
issueSignal ............................................ 187
PIPLIST .................................................. 187
process ................................................... 187
put ......................................................... 188
receive ..................................................... 188
send (1) ..................................................... 188
send (2) ..................................................... 188
sendInvite (1) ............................................ 188
sendInvite (2) ............................................ 189
sendLast (1) ............................................... 189
sendLast (2) ............................................... 189
state ......................................................... 189
stateText .................................................. 190
tsyncLevel ............................................... 190
Inherited public methods .............................. 190
Inherited protected methods ............................ 191
Enumerations ............................................. 191
AllocateOpt .............................................. 191
SendOpt .................................................... 191
StateOpt ................................................... 191
SyncLevel .................................................. 191

Chapter 47. IccStartRequestQ class .............................. 193
IccStartRequestQ constructor (protected) ...................... 193
    Constructor ........................................... 193
Public methods ........................................... 193
cancel ..................................................... 193
clearData ................................................ 193
data ......................................................... 194
instance ............................................... 194
queueName .............................................. 194
registerData .......................................... 194
reset ....................................................... 194
retrieveData ............................................ 194
returnTermId .......................................... 195
returnTransId ......................................... 195
setData .................................................. 195
setQueueName .......................................... 195
setReturnTermId (1) ..................................... 195
setReturnTermId (2) ..................................... 195
setReturnTransId (1) ................................... 195
setReturnTransId (2) ................................... 195
setStartOpts .......................................... 196
start ...................................................... 196
Inherited public methods .............................. 197
Inherited protected methods ............................ 197
Enumerations ............................................. 198
RetrieveOpt .............................................. 198
ProtectOpt ............................................... 198
CheckOpt ............................................... 198
Chapter 48. IccSysId class ................................................. 199
IccSysId constructors ................................................. 199
   Constructor (1) ................................................. 199
   Constructor (2) ................................................. 199
Public methods ....................................................... 199
   operator= (1) .................................................. 199
   operator= (2) .................................................. 199
Inherited public methods ........................................... 199
Inherited protected methods ....................................... 200

Chapter 49. IccSystem class ........................................... 201
IccSystem constructor (protected) ................................. 201
Constructor .......................................................... 201
Public methods ....................................................... 201
   applName ....................................................... 201
   beginBrowse (1) .............................................. 201
   beginBrowse (2) .............................................. 201
   dateFormat ................................................... 202
   endBrowse ..................................................... 202
   freeStorage .................................................... 202
   getFile (1) ..................................................... 202
   getFile (2) ..................................................... 202
   getNextFile ................................................... 203
   getStorage ..................................................... 203
   instance ....................................................... 203
   operatingSystem ............................................. 203
   operatingSystemLevel ....................................... 203
   release ......................................................... 204
   releaseText .................................................... 204
   sysId .......................................................... 204
   workArea ....................................................... 204
Inherited public methods ........................................... 204
Inherited protected methods ....................................... 205
Enumerations .......................................................... 205
   ResourceType .................................................. 205

Chapter 50. IccTask class .............................................. 207
IccTask Constructor (protected) .................................... 207
Constructor .......................................................... 207
Public methods ....................................................... 207
   abend .......................................................... 207
   abendData ..................................................... 207
   commitUOW ..................................................... 207
   delay .......................................................... 208
   dump .......................................................... 208
   enterTrace ..................................................... 208
   facilityType ................................................... 209
   freeStorage .................................................... 209
   getStorage ..................................................... 209
   instance ....................................................... 210
   isCommandSecurityOn ....................................... 210
   isCommitSupported ......................................... 210
   isResourceSecurityOn ..................................... 210
   isRestarted ................................................... 210
   isStartDataAvailable ...................................... 210
   number ......................................................... 211
Inherited protected methods ........................................... 222

Chapter 53. IccTermId class ........................................... 223
IccTermId constructors .................................................. 223
  Constructor (1) ...................................................... 223
  Constructor (2) ...................................................... 223
Public methods ......................................................... 223
  operator= (1) ...................................................... 223
  operator= (2) ...................................................... 223
Inherited public methods .............................................. 223
Inherited protected methods ........................................ 224

Chapter 54. IccTerminal class ......................................... 225
IccTerminal constructor (protected) ................................ 225
  Constructor ......................................................... 225
Public methods ......................................................... 225
  AID ................................................................. 225
  clear ............................................................... 225
  cursor .............................................................. 225
  data ................................................................. 225
  erase ............................................................... 226
  freeKeyboard ...................................................... 226
  get ................................................................. 226
  height ............................................................... 226
  inputCursor ....................................................... 226
  instance ........................................................... 226
  line ................................................................. 226
  netName ............................................................ 227
  operator<< (1) ..................................................... 227
  operator<< (2) ..................................................... 227
  operator<< (3) ..................................................... 227
  operator<< (4) ..................................................... 227
  operator<< (5) ..................................................... 227
  operator<< (6) ..................................................... 227
  operator<< (7) ..................................................... 227
  operator<< (8) ..................................................... 227
  operator<< (9) ..................................................... 227
  operator<< (10) ................................................... 227
  operator<< (11) ................................................. 227
  operator<< (12) ................................................. 227
  operator<< (13) ................................................. 227
  operator<< (14) ................................................. 227
  operator<< (15) ................................................. 227
  operator<< (16) ................................................. 227
  operator<< (17) ................................................. 227
  operator<< (18) ................................................. 227
  put ................................................................. 228
  receive ............................................................ 228
  receive3270Data .................................................. 229
  send (1) ........................................................... 229
  send (2) ........................................................... 229
  send (3) ........................................................... 229
  send (4) ........................................................... 229
  send3270 (1) ...................................................... 230
  send3270 (2) ...................................................... 230
  send3270 (3) ...................................................... 230
send3270 (4) ........................................ 230
sendLine (1) ........................................ 230
sendLine (2) ........................................ 231
sendLine (3) ........................................ 231
sendLine (4) ........................................ 231
setColor .......................................... 231
setCursor (1) ...................................... 231
setCursor (2) ...................................... 232
setHighlight ...................................... 232
setLine .............................................. 232
setNewLine ........................................ 232
setNextCommArea ............................. 232
setNextInputMessage ....................... 233
setNextTransId .............................. 233
signoff ............................................. 233
signon (1) ......................................... 233
signon (2) ......................................... 233
waitForAID (1) .................................. 234
waitForAID (2) .................................. 234
width ............................................... 234
workArea .......................................... 234
Inherited public methods ............................ 234
Inherited protected methods ....................... 235
Enumerations ...................................... 235
AIDVal ............................................. 235
Case ................................................. 235
Color ............................................... 235
Highlight ......................................... 235
NextTransIdOpt .................................. 235

Chapter 55. IccTerminalData class .......................... 237
IccTerminalData constructor (protected) ............... 237
Constructor ....................................... 237
Public methods ................................... 237
alternateHeight .................................. 237
alternateWidth .................................. 237
defaultHeight .................................. 237
defaultWidth .................................... 238
graphicCharCodeSet ............................ 238
graphicCharSetId ............................... 238
isAPLKeyboard ................................... 238
isAPLText ......................................... 238
isBTrans .......................................... 238
isColor ............................................. 239
isEWA .............................................. 239
isExtended3270 ................................. 239
isFieldOutline .................................. 239
isGoodMorning .................................. 239
isHighlight ...................................... 240
isKatakana ....................................... 240
isMSRControl .................................... 240
isPS ............................................... 240
isSOSI ............................................. 240
isTextKeyboard .................................. 240
isTextPrint ....................................... 241
isValidation ...................................... 241
Inherited public methods ........................................ 241
Inherited protected methods .................................. 241

Chapter 56. IccTime class ........................................ 243
IccTime constructor (protected) ................................. 243
  Constructor .................................................. 243
Public methods .................................................. 243
  hours ......................................................... 243
  minutes ...................................................... 243
  seconds ...................................................... 243
  timeInHours ............................................... 243
  timeInMinutes ............................................ 244
  timeInSeconds ............................................ 244
  type ......................................................... 244
Inherited public methods ...................................... 244
Inherited protected methods .................................. 244
Enumerations .................................................... 244
  Type .......................................................... 244

Chapter 57. IccTimeInterval class ............................. 247
IccTimeInterval constructors .................................... 247
  Constructor (1) ............................................. 247
  Constructor (2) ............................................. 247
Public methods .................................................. 247
  operator= .................................................. 247
  set ........................................................ 247
Inherited public methods ...................................... 248
Inherited protected methods .................................. 248

Chapter 58. IccTimeOfDay class ................................. 249
IccTimeOfDay constructors ...................................... 249
  Constructor (1) ............................................. 249
  Constructor (2) ............................................. 249
Public methods .................................................. 249
  operator= .................................................. 249
  set ........................................................ 249
Inherited public methods ...................................... 250
Inherited protected methods .................................. 250

Chapter 59. IccTPNameId class ................................. 251
IccTPNameId constructors ...................................... 251
  Constructor (1) ............................................. 251
  Constructor (2) ............................................. 251
Public methods .................................................. 251
  operator= (1) .............................................. 251
  operator= (2) .............................................. 251
Inherited public methods ...................................... 251
Inherited protected methods .................................. 252

Chapter 60. IccTransId class ................................. 253
IccTransId constructors ...................................... 253
  Constructor (1) ............................................. 253
  Constructor (2) ............................................. 253
Public methods .................................................. 253
  operator= (1) .............................................. 253
  operator= (2) .............................................. 253
## Chapter 61. IccUser class

### IccUser constructors
- Constructor (1)  
- Constructor (2)

### Public methods
- changePassword
- daysUntilPasswordExpires
- ESMReason
- ESMResponse
- groupId
- invalidPasswordAttempts
- language
- lastPasswordChange
- lastUseTime
- passwordExpiration
- setLanguage
- verifyPassword

## Chapter 62. IccUserId class

### IccUserId constructors
- Constructor (1)  
- Constructor (2)

### Public methods
- operator= (1)  
- operator= (2)

## Chapter 63. IccValue structure

### Enumeration
- CVDA

## Chapter 64. main function
First Screen ........................................... 284
Second Screen ........................................... 284
ICC$RES1 (IRE1) ......................................... 284
ICC$RES2 (IRE2) ......................................... 285
ICC$SEM (ISEM) .......................................... 285
ICC$SES1 (ISE1) .......................................... 285
ICC$SES2 (ISE2) .......................................... 286
ICC$SRQ1 (ISR1) .......................................... 286
ICC$SRQ2 (ISR2) .......................................... 286
ICC$SYS (ISYS) ........................................... 287
ICC$TMP (ITMP) ........................................... 287
ICC$TRM (ITRM) ........................................... 287
ICC$TSK (ITSK) ........................................... 288

Glossary .................................................. 289

Bibliography ............................................. 291
CICS Transaction Server for z/OS ..................... 291
  CICS books for CICS Transaction Server for z/OS .... 291
  CICSPlex SM books for CICS Transaction Server for z/OS 292
  Other CICS books ....................................... 292
Related books ............................................ 292
  C++ Programming ....................................... 292
  CICS client manuals .................................... 293
Determining if a publication is current ............... 293

Index ..................................................... 295

Notices ................................................... 333
Trademarks .............................................. 334

Sending your comments to IBM ......................... 335
Preface

The CICS® family provides robust transaction processing capabilities across the major hardware platforms that IBM® offers, and also across key non-IBM platforms. It offers a wide range of features for supporting client/server applications, and allows the use of modern graphical interfaces for presenting information to the end-user. The CICS family now supports the emerging technology for object oriented programming and offers CICS users a way of capitalizing on many of the benefits of object technology while making use of their investment in CICS skills, data and applications.

Object oriented programming allows more realistic models to be built in flexible programming languages that allow you to define new types or classes of objects, as well as employing a variety of structures to represent these objects.

Object oriented programming also allows you to create methods (member functions) that define the behavior associated with objects of a certain type, capturing more of the meaning of the underlying data.

The CICS foundation classes software is a set of facilities that IBM has added to CICS to make it easier for application programmers to develop object oriented programs. It is not intended to be a product in its own right.

The CICS C++ foundation classes, as described here, allow an application programmer to access many of the CICS services that are available via the EXEC CICS procedural application programming interface (API). They also provide an object model, making OO application development simpler and more intuitive.

Who this book is for

This book is for CICS application programmers who want to know how to use the CICS foundation classes.

What this book is about

This book is divided into three parts and three appendixes:

- **Part 1. Installation and setup** describes how to install the product and check that the installation is complete.
- **Part 2. Using the CICS foundation classes** describes the classes and how to use them.
- **Part 3. Foundation Classes—reference** contains the reference material: the class descriptions and their methods.
- For those of you familiar with the EXEC CICS calls, [Appendix A. Mapping EXEC CICS calls to Foundation Class methods](#) maps EXEC CICS calls to the foundation class methods detailed in this book...
- ... and [Appendix B. Mapping Foundation Class methods to EXEC CICS calls](#) maps them the other way — foundation class methods to EXEC CICS calls.
- [Appendix C. Output from sample programs](#) contains the output from the sample programs.
What you need to know before reading this book

Chapter 1. Getting ready for object oriented CICS on page 3 describes what you need to know to understand this book.

Notes on terminology

“CICS” is used throughout this book to mean the CICS element of the IBM CICS Transaction Server for z/OS, Version 2 Release 1.

“RACF® ” is used throughout this book to mean the MVS™ Resource Access Control Facility (RACF) or any other external security manager that provides equivalent function.

In the programming examples in this book, the dollar symbol ($) is used as a national currency symbol. In countries where the dollar is not the national currency, the local currency symbol should be used.
Part 1. Installation and setup

This part of the book describes the CICS foundation classes installed on your CICS server.
Chapter 1. Getting ready for object oriented CICS

This book makes several assumptions about you, the reader. It assumes you are familiar with:

- Object oriented concepts and technology
- C++ language
- CICS.

This book is not intended to be an introduction to any of these subjects. If the terms in the [Glossary on page 289](#) are not familiar to you, then please consult other sources before going any further. A selection of appropriate books may be found in the bibliography on page [Bibliography on page 291](#), but you may find other books useful too.
Chapter 2. Installed contents

The CICS foundation classes package consists of several files or datasets. 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.

<table>
<thead>
<tr>
<th>C++ Header File</th>
<th>Classes Defined in this Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICCABDEH</td>
<td>lccAbendData</td>
</tr>
<tr>
<td>ICCBASEH</td>
<td>lccBase</td>
</tr>
<tr>
<td>ICCBUFEH</td>
<td>lccBuf</td>
</tr>
<tr>
<td>ICCCLKEH</td>
<td>lccClock</td>
</tr>
<tr>
<td>ICCCNDEH</td>
<td>lccCondition (struct)</td>
</tr>
<tr>
<td>ICCCONEH</td>
<td>lccConsole</td>
</tr>
<tr>
<td>ICCCTLEH</td>
<td>lccControl</td>
</tr>
<tr>
<td>ICCDATEH</td>
<td>lccDataQueue</td>
</tr>
<tr>
<td>ICCEH</td>
<td>see [1]</td>
</tr>
<tr>
<td>ICCEVTIEH</td>
<td>lccEvent</td>
</tr>
<tr>
<td>ICCEXCEH</td>
<td>lccException</td>
</tr>
<tr>
<td>ICCFILEH</td>
<td>lccFile</td>
</tr>
<tr>
<td>ICCFILIEH</td>
<td>lccFileIterator</td>
</tr>
<tr>
<td>ICCGLBEH</td>
<td>lcc (struct) (global functions)</td>
</tr>
<tr>
<td>ICCJRNIEH</td>
<td>lccJournal</td>
</tr>
<tr>
<td>ICCMSEGEH</td>
<td>lccMessage</td>
</tr>
<tr>
<td>ICCPRGEH</td>
<td>lccProgram</td>
</tr>
<tr>
<td>ICCRECIEH</td>
<td>lccRecordIndex, lccKey, lccRBA and lccRRN</td>
</tr>
<tr>
<td>ICCRESEH</td>
<td>lccResource</td>
</tr>
<tr>
<td>ICCRIDEH</td>
<td>lccResourceld + subclasses (such as lccConvId)</td>
</tr>
<tr>
<td>ICCSEMIEH</td>
<td>lccSemaphore</td>
</tr>
<tr>
<td>ICCSEEH</td>
<td>lccSession</td>
</tr>
<tr>
<td>ICCSRQEH</td>
<td>lccStartRequestQ</td>
</tr>
<tr>
<td>ICCSYSEH</td>
<td>lccSystem</td>
</tr>
<tr>
<td>ICCTIMEH</td>
<td>lccTime, lccAbsTime, lccTimeInterval, lccTimeOfDay</td>
</tr>
<tr>
<td>ICTXMDEH</td>
<td>lccTerminalData</td>
</tr>
<tr>
<td>ICCTMPHEH</td>
<td>lccTempStore</td>
</tr>
<tr>
<td>ICCTRMEH</td>
<td>lccTerminal</td>
</tr>
<tr>
<td>ICCTSKEH</td>
<td>lccTask</td>
</tr>
<tr>
<td>ICCUSREH</td>
<td>lccUser</td>
</tr>
<tr>
<td>ICCVALEH</td>
<td>lccValue (struct)</td>
</tr>
</tbody>
</table>

Notes:

1. A single header that #includes all the above header files is supplied as ICCEH

© Copyright IBM Corp. 1989, 2001
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.

Location
PDS: CICSTS21.CICS.SDFHC370

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

Location
ICCFCDLL module in PDS: CICSTS21.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: CICSTS21.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
CICSTS21.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 dataset
CICSTS21.CICS.SDFHPROC.

ICCFCL 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" on page 45.

Sample programs such as ICC$BUF, ICC$CLK 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 datasets for CICS Transaction Server for z/OS
CICSTS21.CICS.SDFHSOCK contains the member
ICCFCImp - 'sidedeck' containing import control statements

CICSTS21.CICS.SDFHPROC contains the members
ICCFCC - JCL to compile a CFC user program
ICCFCL - JCL to compile, prelink and link a CFC user program
ICCFCL - JCL to prelink and link a CFC user program
CICSTS21.CICS.SDFHLOAD contains the members
  DFHCURDS - program definitions required for CICS system definition.
  DFHCURDI - program definitions required for CICS system definition.
Installed contents
Chapter 3. Hello World

When you start programming in an unaccustomed environment the hardest task is usually getting something—anything—to work and to be seen to be working. The initial difficulty is not in the internals of the program, but in bringing everything together—the CICS server, the programming environment, program inputs and program outputs.

The example shown in this chapter shows how to get started in CICS OO programming. It is intended as an appetizer; Chapter 5. Overview of the foundation classes on page 14 is a more formal introduction and you should read it before you attempt serious OO programming.

This example could not be much simpler but when it works it is a visible demonstration that you have got everything together and can go on to greater things. The program writes a simple message to the CICS terminal.

There follows a series of program fragments interspersed with commentary. The source for this program can be found in sample ICC$HEL (see Sample source code on page 6 for the location).

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

void IccUserControl::run()
{
    IccTerminal* pTerm = terminal();
    pTerm->erase();
    pTerm->send(10, 35, "Hello World");
}
```

The terminal method of IccControl class is used to obtain a pointer to the terminal object for the application to use.

```c
pTerm->erase();
```

The erase method clears the current contents of the terminal.

```c
pTerm->send(10, 35, "Hello World");
```
Hello World

The `send` method is called on the terminal object. This causes "Hello World" to be written to the terminal screen, starting at row 10, column 35.

```cpp
pTerm->waitForAID();
```

This waits until the terminal user hits an AID (Action Identifier) key.

```cpp
return;
}
```

Returning from the `run` method causes program control to return to CICS.

Compile and link "Hello World"

The "Hello World" sample is provided as sample ICC$HEL (see [Sample source code" on page 6). Find this sample and copy it to your own work area.

To compile and link any CICS C++ Foundation program you need access to:
1. The source of the program, here ICC$HEL.
2. The Foundation Classes header files (see [Header files" on page 5).
3. The Foundation Classes dynamic link library (see [Dynamic link library" on page 6).

See [Chapter 8. Compiling, executing, and debugging" on page 45 for the JCL required to compile the sample program.

Running "Hello World" on your CICS server

To run the program you have just compiled on your CICS server, you need to make the executable program available to CICS (that is, make sure it is in a suitable directory or load library). Then, depending on your server, you may need to create a CICS program definition for your executable. Finally, you may logon to a CICS terminal and run the program.

To do this,
1. Logon to a CICS terminal and enter either:
   ```
   IHEL
   ```
   or
   ```
   CECI LINK PROGRAM(ICC$HEL)
   ```
2. If you are not using program autoinstall on your CICS region, define the program ICC$HEL to CICS using the supplied transaction CEDA.
3. Log on to a CICS terminal.
4. On CICS terminal run:
   ```
   CECI LINK PROGRAM(ICC$HEL)
   ```

Expected Output from "Hello World"

This is what you should see on the CICS terminal if program ICC$HEL has been successfully built and executed.

```
Hello World
```

Hit an Action Identifier, such as the ENTER key, to return.
This part of the book describes the CICS foundation classes and how to use them. There is a formal listing of the user interface in "Part 3. Foundation Classes—reference" on page 61.
Chapter 4. C++ Objects

This chapter 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:

```cpp
{  
  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:

```cpp
{  
  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 above 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:

```cpp
{  
  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,
**C++ 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 57.

---

**Using an object**

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:

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

Alternatively, you can do this using dynamic object creation:

```cpp
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 automatically 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.
This chapter is a formal introduction to what the Foundation Classes can do for you. See [Chapter 3. Hello World](#) on page 9 for a simple example to get you started.

The chapter takes a brief look at the CICS C++ Foundation Class library by considering the following categories in turn:

- "Base classes"
- "Resource identification classes" on page 16
- "Resource classes" on page 17
- "Support Classes" on page 18

See [Part 3. Foundation Classes—reference](#) on page 61 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 17.

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

© Copyright IBM Corp. 1989, 2001
Base classes

**IccTime**

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

### Resource identification classes

CICS resource identification classes define CICS resource identifiers – typically entries in one of the CICS tables. For example an **IccFileId** object represents a CICS file name – an FCT (file control table) entry. 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 table entry 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 17, 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.

<table>
<thead>
<tr>
<th>Class</th>
<th>CICS resource</th>
<th>CICS table</th>
</tr>
</thead>
<tbody>
<tr>
<td>IccAlarmRequestId</td>
<td>alarm request</td>
<td></td>
</tr>
<tr>
<td>IccConvId</td>
<td>conversation</td>
<td></td>
</tr>
<tr>
<td>IccDataQueueId</td>
<td>data queue</td>
<td>DCT</td>
</tr>
</tbody>
</table>

---

*Figure 2. Resource identification classes*
Resource identification classes

<table>
<thead>
<tr>
<th>Class</th>
<th>CICS resource</th>
<th>CICS table</th>
</tr>
</thead>
<tbody>
<tr>
<td>IccFileld</td>
<td>file</td>
<td>FCT</td>
</tr>
<tr>
<td>IccGroupId</td>
<td>group</td>
<td></td>
</tr>
<tr>
<td>IccJournalld</td>
<td>journal</td>
<td>JCT</td>
</tr>
<tr>
<td>IccJournalTypeld</td>
<td>journal type</td>
<td></td>
</tr>
<tr>
<td>IccLockld</td>
<td>(Not applicable)</td>
<td></td>
</tr>
<tr>
<td>IccPartnerId</td>
<td>APPC partner definition files</td>
<td></td>
</tr>
<tr>
<td>IccProgramId</td>
<td>program</td>
<td>PPT</td>
</tr>
<tr>
<td>IccRequestId</td>
<td>request</td>
<td></td>
</tr>
<tr>
<td>IccSysId</td>
<td>remote system</td>
<td>TST</td>
</tr>
<tr>
<td>IccTempStoreId</td>
<td>temporary storage</td>
<td>TCT</td>
</tr>
<tr>
<td>IccTermId</td>
<td>terminal</td>
<td></td>
</tr>
<tr>
<td>IccTPNameId</td>
<td>remote APPC TP name</td>
<td></td>
</tr>
<tr>
<td>IccTransId</td>
<td>transaction</td>
<td>PCT</td>
</tr>
<tr>
<td>IccUserId</td>
<td>user</td>
<td>SNT</td>
</tr>
</tbody>
</table>

Resource classes

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

**LccBase**

- **LccResource**
  - LccAbendData
  - LccClock
  - LccConsole
  - LccControl
  - LccDataQueue
  - LccFile
  - LccFileIterator
  - LccJournal
  - LccProgram
  - LccSemaphore
  - LccSession
  - LccStartRequestQ
  - LccSystem
  - LccTask
  - LccTempStore
  - LccTerminal
  - LccTerminalData
  - LccUser

*Figure 3. Resource classes*

- Terminals are modelled by LccTerminal.
- Programs are modelled by LccProgram.
- Temporary Storage queues are modelled by LccTempStore.
- Transient Data queues are modelled by LccDataQueue.

All CICS resource classes inherit from the LccResource base class. For example, any operation on a CICS resource may raise a CICS condition; the condition method of LccResource (see page 170) can interrogate it.
Resource classes

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

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

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.

IccBase

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

Figure 4. Support classes

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

CICS Transaction Server: C++ OO Class Libraries
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:

```cpp
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:

```cpp
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:

```cpp
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 "Chapter 6. Buffer objects" on page 23).

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 simply the application’s view of the resource. The same is true of destroying objects.

You are recommended to use an accompanying resource identification object when creating a resource object. For example:
Using CICS resources

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

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

```c++
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:

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

**Singleton classes**

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

```c++
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:

```c++
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:

```c++
IccTempStoreId id("TEMP1234");
IccTempStore temp(id);
temp.writeItem("Hello TEMP1234");
```
Using CICS resources

Method **writeItem** writes the contents of the string it is passed ("Hello TEMP1234") to the CICS Temporary Storage queue "TEMP1234".
Chapter 6. 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);
```
Buffer objects

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 above 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 this, consider writing a queue item to CICS temporary storage using IccTempstore class.
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').

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

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.

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

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.

```
buffer = queue.readItem();
buffer << "Some extra data";
queue.writeItem(buffer);
```

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

Please refer to the reference section for further information on the IccBuf class. You might also find the supplied sample — ICC$BUF — helpful.
Chapter 7. Using CICS Services

This chapter describes how to use CICS services. The following services are considered in turn:

- "File control"
- "Program control" on page 32
- "Starting transactions asynchronously" on page 34
- "Transient Data" on page 37
- "Temporary storage" on page 39
- "Terminal control" on page 41
- "Time and date services" on page 43

File control

The file control classes – lccFile, lccFileld, lccKey, lccRBA, and lccRRN – allow you to read, write, update and delete records in files. In addition, lccFileiterator class allows you to browse through all the records in a file.

An lccFile object is used to represent a file. It is convenient, but not necessary, to use an lccFileld 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 – lccFile to perform the operation and one of lccKey, lccRBA, and lccRRN to identify the particular record, depending on whether the file access type is KSDS, ESDS, or RRDS.
The `readRecord` method of `IccFile` class actually 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 section 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 29 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 actually 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 \texttt{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 \texttt{writeRecord} method of \texttt{IccFile} class actually writes the record.

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

Writing ESDS records
You must use a relative byte address (RBA) held in an \texttt{IccRBA} object to access the beginning of a record.

\texttt{IccFile} class has methods \texttt{isAddable}, \texttt{recordFormat}, \texttt{recordIndex}, \texttt{recordLength}, and \texttt{registerRecordIndex} that help you when writing ESDS records.

Writing RRDS records
Use the \texttt{writeRecord} method to add a new ESDS record. After writing the record you can use the \texttt{number} method on the \texttt{IccRBA} object to discover the assigned relative byte address for the record you have just written.

\texttt{IccFile} class has methods \texttt{isAddable}, \texttt{recordFormat}, \texttt{recordIndex}, \texttt{recordLength}, and \texttt{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 \texttt{readRecord} method in 'update' mode. This locks the record so that nobody else can change it.

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

For an example, see \texttt{"code fragment: "Read_record_for_update"" on page 32}.

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 \texttt{deleteRecord} method of \texttt{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 \texttt{IccKey} or \texttt{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" on page 31].

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 the samples directory (see [Sample source code on page 6]) 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.

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

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.

```c
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.
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.

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

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'.

```c
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);
}
```

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).

```c
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 actually 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.

```c
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.
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.

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

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

```cpp
return;
}
```

The end of run, which returns control to CICS.

See "Appendix C. Output from sample programs" on page 281 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 "Chapter 37. IccProgram class" on page 157 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, in the samples directory (see "Sample source code" on page 6).

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

```cpp
#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.

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

See [Appendix C. Output from sample programs: on page 281][1] for the expected output from these sample programs.

## 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

<table>
<thead>
<tr>
<th>CICS system</th>
<th>ICC1</th>
<th>ICC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction</td>
<td>ISR1/ITMP</td>
<td>ISR2</td>
</tr>
<tr>
<td>Program</td>
<td>ICC$SRQ1/ICC$TMP</td>
<td>ICC$SRQ2</td>
</tr>
<tr>
<td>Terminal</td>
<td>PEO1</td>
<td>PEO2</td>
</tr>
</tbody>
</table>

The scenario is as follows. We start transaction ISR1 on terminal PEO1 on system ICC1. 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).

[1]: #/output-from-sample-programs-on-page-281
The programs can be found in the samples directory (see [Sample source code] on page 6) 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:

```c++
#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.

```c++
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.

```c++
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 actually 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.
Starting transactions asynchronously

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.
Starting transactions asynchronously

```
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.

See "Appendix C. Output from sample programs" on page 281 for the expected output from these sample programs.

### 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 35.

### 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 in the samples directory (see "Sample source code" on page 6) as file ICC$DAT. Here the code is presented without the terminal IO requests.

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

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

void IccUserControl::run()
{

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

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

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

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

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

The end of run, which returns control to CICS.

See [Appendix C. Output from sample programs on page 281](#) for the expected output from this sample program.

### 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 37](#).

#### 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 40](#) 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.
Temporary storage

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 can be found in the samples directory (see "Sample source code" on page 6) as file ICC$TMP. The sample is presented here without the terminal IO requests.

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

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

void IccUserControl::run()
{
    short itemNum = 1;
    IccTempStoreId id("ICCSTORE");
    IccTempStore store( id );
    IccBuf buffer( 50 );
    store.empty();

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

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

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.

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 );
}

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

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.

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.

See [Appendix C. Output from sample programs on page 281] for the expected output from this sample program.

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. Alternatively, you can use the "<<" operators to send data to the terminal.

Before sending data to a terminal, you may want to set, for example, the position of the cursor on the screen or the color of the text. 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.
Terminal control

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 described 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 can be found in the samples directory (see [Sample source code” on page 6]) as file ICC$TRM.

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

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

  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" );
  term.setNewLine();
}
```

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*`).

```cpp
term.sendLine();
```

This sends a blank line to the screen.
The `setColor` method is used to set the colour of the text on the screen and the `setHighlight` method to set the highlighting.

```cpp
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 colour of the text on the screen and the `setHighlight` method to set the highlighting.

```cpp
term << "A cout style 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.

```cpp
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.

```cpp
return;
}
```

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

See [Appendix C. Output from sample programs on page 281] for the expected output from this sample program.

---

### 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 can be found in the samples directory (see "Sample source code" on page 6) as file `ICCS$CLK`. The sample is presented here without the terminal IO requests.

```cpp
#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.
Time and date services

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.

See [Appendix C. Output from sample programs on page 281] for the expected output from this sample program.
Chapter 8. Compiling, executing, and debugging

This chapter describes how to compile, execute, and debug a CICS Foundation Class program. The following are considered in turn:

- **Compiling Programs**
- **Executing Programs**
- **Debugging Programs**

### Compiling Programs

To compile and link a CICS Foundation Class program you need access to the following:

- 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 5)
  - The Foundation Classes dynamic link library (DLL) (see [Dynamic link library](#) on page 6)

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:

```c
//ICC$HEL JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//PROCLIB JCLLIB ORDER=(CICSTS21.CICS.SDFHPROC)
//ICC$HEL EXEC ICCFCCL,INFILE=indatasetname(ICC$HEL),OUTFILE=outdatasetname(ICC$HEL)
```

### 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.

### Debugging Programs

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

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

1. Use a symbolic debugger
Compiling, executing, and debugging

2. Run the Foundation Class Program with tracing active
3. Run the Foundation Class Program with the CICS Execution Diagnostic Facility

Symbolic Debuggers

A symbolic debugger allows you to step through the source of your CICS Foundation Classes program. **Debug Tool**, a component of CODE/370, is shipped as a feature with IBM C/C++ for OS/390.

To debug a CICS Foundation Classes program with a symbolic debugger, you need to compile the program with a flag that adds debugging information to your executable. For CICS Transaction Server for z/OS, this is TEST(ALL).

For more information see [Debug Tool User’s Guide and Reference, SC09-2137](#).

Tracing a Foundation Class Program

The CICS Foundation Classes can be configured to write a trace file for debugging/service purposes.

**Activating the trace output**

In CICS Transaction Server for z/OS, exception trace 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

For the EXEC CICS API, there is a CICS facility called the Execution Diagnostic Facility (EDF) that allows you to step through your CICS program stopping at each EXEC CICS call. This does not make much sense from the CICS Foundation Classes because the display screen shows the procedural EXEC CICS call interface rather than the CICS Foundation Class type interface. However, this may be of use to programmers familiar with the EXEC CICS interface.

**Enabling EDF**

To enable EDF, use the pre-processor macro **ICC_EDF** – this can be done in your source code before including the file ICCMAIN as follows:

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

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

For more information about using EDF see “Execution diagnostic facility (EDF)” in [CICS Application Programming Guide](#).
Chapter 9. Conditions, errors, and exceptions

This chapter describes how the Foundation Classes have been designed to respond to various error situations they might encounter. These will be discussed under the following headings:

- "Foundation Class Abend codes"
- "C++ Exceptions and the Foundation Classes"
- "CICS conditions" on page 49
- "Platform differences" on page 52

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 and Codes, GC34-5716.

C++ Exceptions and the Foundation Classes

C++ exceptions are managed using the reserved words try, throw, and catch. Please refer to your compiler's documentation or one of the C++ books in the bibliography for more information.

Here is sample ICC$EXC1 (see "Sample source code" on page 6):

```cpp
#include "icceh.hpp"
#include "icccmain.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.
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!!
```

The CICS C++ Foundation Classes do not throw char* exceptions as in the above 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,
Conditions, errors, exceptions

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 the samples directory (see "Sample source code" on page 6) as file ICC$EXC2. The sample is presented here without many of the terminal IO requests.

```cpp
#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 above 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 "Appendix C. Output from sample programs" on page 281 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 "Part 3. Foundation Classes—reference" on page 61.

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

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). It can be explicitly activated as follows:

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

This setting means that when CICS raises the QIDERR condition as a result of action on the ‘temp’ object, no action is taken. This means that the condition must be handled manually, using the `condition` method. For example:

```cpp
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:

```cpp
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.

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

Now implement the handleEvent method.

```cpp
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:

```cpp
IccTempStore temp("TEMPl234");
temp.setActionOnCondition(IccResource::throwException,
IccCondition::QIDERR);
```

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

```cpp
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...
Exception handling

and resumes execution at an appropriate catch block – it is not possible to resume within the try block. For a fuller example of the above, 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 47 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

Note: References in this section to other CICS platforms—CICS OS/2 and CICS for AIX—are included for completeness. There have been Technology Releases of the CICS Foundation Classes on those platforms.

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.

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 (MVS, UNIX, and OS/2).

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 [Part 3 Foundation Classes—reference on page 61]. Platform variations exist at:

- object level
- method level
- parameter level

Object level

Some objects are not supported on certain platforms. For example IccJournal objects cannot be created on CICS OS/2 as CICS OS/2 does not support journalling services. IccConsole objects cannot be created on CICS for AIX as CICS for AIX does not support console services.
Any attempt to create **IccJournal** on CICS OS/2, or an **IccConsole** object on CICS for AIX causes an **IccException** object of type 'platformError' to be thrown, but would be acceptable on the other platforms.

For example:

```cpp
IccJournal journal7(7); //No good on CICS OS/2
```

or

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

If you initialize your application with 'fsEnforce' selected (see "initializeEnvironment" on page 63) 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. [Part 3, Foundation Classes—reference on page 61] fully documents all platform restrictions.

### Method level

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

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

Here method **programId** executes correctly on CICS OS/2 and CICS/ESA but throws an **IccException** object of type 'platformError' on CICS for AIX.

Alternatively, if you initialize your application with family subset enforcement on (see **initializeEnvironment** function of **Icc** structure) then 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.

```cpp
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 for AIX platform, but not on a CICS OS/2 or CICS/ESA platform.
If family subset enforcement is on, abend 5 throws an IccException object of type ‘familyConformanceError’, irrespective of the target CICS platform.
Chapter 10. Miscellaneous

This chapter describes the following:
- “Polymorphic Behavior”
- “Storage management” on page 57
- “Parameter passing conventions” on page 58
- “Scope of data in IccBuf reference returned from ‘read’ methods” on page 59

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:

```
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 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**:

```cpp
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:

<table>
<thead>
<tr>
<th>Class</th>
<th>clear</th>
<th>get</th>
<th>put</th>
</tr>
</thead>
<tbody>
<tr>
<td>IccConsole</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>IccDataQueue</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IccJournal</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>IccSession</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IccTempStore</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IccTerminal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

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

**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 (see “Sample source code” on page 6) as file ICCRES2. It is presented here without the terminal IO requests.

```cpp
#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.

```cpp
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.

```cpp
pObj[0] = new IccDataQueue("ICCQ");
pObj[1] = new IccTempStore("ICCTEMPS");
```
We create two objects whose classes are derived from `IccResource` – `IccDataQueue` and `IccTempStore`.

```cpp
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.

```cpp
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.

```cpp
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— see [Creating an object](#) on page 13. 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 OS/390, CICS and Language Environment® manage all task storage so that it is released at task termination (normal or abnormal).

In a CICS for OS/2® or CICS for AIX environment, as in the earlier Technology Releases for those platforms, 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 \textbf{new} and \textbf{delete} in the base Foundation Class, \texttt{IccBase}. These can be configured to map dynamic storage allocation requests to CICS task storage, so that \textit{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 \texttt{Icc::initializeEnvironment} call that must be issued before using the Foundation Classes. (This function is called from the default main function—see \textit{Chapter 64, main function} on page 265.)

The first parameter passed to the \texttt{initializeEnvironment} function is an enumeration that takes one of these three values:

- \texttt{cmmDefault}
  - The default action is platform dependent:
    - MVS/ESA™: same as 'cmmNonCICS' - see below.
    - UNIX: same as 'cmmCICS' - see below.
    - OS/2: same as 'cmmCICS' - see below.

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

- \texttt{cmmCICS}
  - The \texttt{new} and \texttt{delete} operators in class \texttt{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 \texttt{initializeEnvironment} with an enum of 'cmmDefault'. You can change this in your program without changing the supplied "header file" ICCMAIN as follows:

\begin{verbatim}
#define ICC_CLASS_MEMORY_MGMT Icc::cmmNonCICS
#include "iccmain.hpp"
\end{verbatim}

Alternatively, set the option \texttt{DEV(ICC\_CLASS\_MEMORY\_MGMT)} when compiling.

\section*{Parameter passing conventions}

The convention used for passing objects on Foundation Classes method calls is as follows:

If the object is mandatory, pass by reference; if it is optional pass by pointer.

For example, consider method \texttt{start} of class \texttt{IccStartRequestQ}, which has the following signature:

\begin{verbatim}
const IccRequestId& start( const IccTransId& transId,
                          const IccTime* time=0,
                          const IccRequestId* reqId=0 );
\end{verbatim}
Using the above 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:

```cpp
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

```cpp
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

```cpp
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 above example).
- The resource updates are committed (see method `IccTask::commitUOW`).
- The task ends (normally or abnormally).
Miscellaneous
This part 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`. You are advised not to 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 on page 5](#).
4. Sample program demonstrating class. For the location of the supplied C++ sample programs on your system see [Sample source code on page 6](#).
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 `IccJournalTypeld`
6. Class and structure names and enumeration types begin with capital letters. Other names begin with lower case letters.

For further information on how to use these classes, see [Part 2. Using the CICS foundation classes on page 11](#).
Chapter 11. 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

**boolText**

```cpp
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.

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

**catchException**

```cpp
static void catchException(IccException& exception)
```

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

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 [Chapter 64. main function](#) on page 265. All OO CICS programs should use this stub or a close equivalent.

**conditionText**

```cpp
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.

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.

**initializeEnvironment**

```cpp
static void initializeEnvironment (ClassMemoryMgmt mem = cmmDefault,
    FamilySubset fam = fsDefault,
    Icc::Bool EDF)
```
An enumeration, defined in this structure, that indicates the memory management policy for the foundation classes.

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

A boolean that indicates whether EDF tracing is initially on. 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 Chapter 64, main function” on page 265. All OO CICS programs should use this stub or a close equivalent.

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

Returns a Boolean value, defined in this structure, that indicates whether EDF tracing is on at the global level. (See setEDF in this structure, isEDFOn and setEDF in IccResource class on page 169 and “Execution Diagnostic Facility” on page 46).

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.

This call returns the program flow to CICS. It is called by the main function in the stub program, listed in ICCMAIN header file, and described in Chapter 64, main function” on page 265. All OO CICS programs should use this stub or a close equivalent.

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.
Sets EDF tracing on or off at the global level.

unknownException

```cpp
static void unknownException()
```

This function is called by the `main` function in `ICCMAIN` header file (see Chapter 64 on page 265) and is used to intercept unknown exceptions. (See also `catchException` in this structure).

Enumerations

**Note:** References in this section to other CICS platforms—CICS OS/2 and CICS 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:
- `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:

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

**BoolSet**

- `trueFalse`
- `yesNo`
- `onOff`

**ClassMemoryMgmt**

**cmmDefault**

The defaults for the different platforms are:

- **MVS/ESA**
  - `cmmNonCICS`
- **OS/2**
  - `cmmCICS`
- **UNIX**
  - `cmmCICS`

**cmmNonCICS**

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

In MVS/ESA LE (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 or CICS for OS/2 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

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 MVS/ESA).

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
Chapter 12. IccAbendData class

IccBase
   IccResource
      IccAbendData

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

Header file: ICCABDEH

IccAbendData constructor (protected)

Constructor

IccAbendData()

Public methods

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

abendCode

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 on page GetOpt.

Returns the current 4-character abend code.

Conditions
   INVREQ

ASRAInterrupt

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

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.

Conditions
   INVREQ
ASRAKeyType

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

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.

Conditions
INVREQ

ASRAPSW

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

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.

Conditions
INVREQ

ASRARegisters

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

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.

Conditions
INVREQ
ASRASpaceType

\texttt{IccValue::CVDA ASRASpaceType(Icc::GetOpt \emph{opt} = Icc::object)}

Returns an enumeration, defined in \texttt{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.

**Conditions**
INVREQ

ASRAStorageType

\texttt{IccValue::CVDA ASRAStorageType(Icc::GetOpt \emph{opt} = Icc::object)}

Returns an enumeration, defined in \texttt{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.

**Conditions**
INVREQ
IccAbendData

instance

    static IccAbendData* instance()

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

isDumpAvailable

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

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.

Conditions
INVREQ

originalAbendCode

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

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

Conditions
INVREQ

programName

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

Returns the name of the program that caused the abend.

Conditions
INVREQ

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
</tbody>
</table>
### IccAbendData

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
IccAbendData
Chapter 13. IccAbsTime class

IccBase
   IccResource
      IccTime
         IccAbsTime

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

Header file: ICCTIMEH

IccAbsTime constructor

Constructor (1)

IccAbsTime(const char* absTime)

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

Constructor (2)

IccAbsTime(const IccAbsTime& time)

The copy constructor.

Public methods

date

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.

Returns the date, as a character string.

Conditions
   INVREQ

dayOfMonth

unsigned long dayOfMonth()

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

Conditions
INVREQ

dayOfWeek

IccClock::DayOfWeek dayOfWeek()
Returns an enumeration, defined in IccClock class, that indicates the day of the week.

Conditions
INVREQ

daysSince1900

unsigned long daysSince1900()
Returns the number of days that have elapsed since the first day of 1900.

Conditions
INVREQ

hours

virtual unsigned long hours() const
Returns the hours component of the time.

milliSeconds

long double milliSeconds()
Returns the number of milliseconds that have elapsed since the first day of 1900.

minutes

virtual unsigned long minutes() const
Returns the minutes component of the time.

monthOfYear

IccClock::MonthOfYear monthOfYear()
Returns an enumeration, defined in IccClock class, that indicates the month of the year.

Conditions
INVREQ

operator=

IccAbsTime& operator=(const IccAbsTime& absTime)
Assigns one IccAbsTime object to another.
packedDecimal

    const char* packedDecimal() const

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

seconds

    virtual unsigned long seconds() const

    Returns the seconds component of the time.

time

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

    timeSeparator
    The character that delimits the time fields. The default is no time separation
character.
    Returns the time as a text string.

    Conditions
    INVREQ

timeInHours

    unsigned long timeInHours()

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

timeInMinutes

    unsigned long timeInMinutes()

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

timeInSeconds

    unsigned long timeInSeconds()

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

year

    unsigned long year()

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

    Conditions
    INVREQ
### Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>hours</td>
<td>IccTime</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>minutes</td>
<td>IccTime</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
<tr>
<td>timeInHours</td>
<td>IccTime</td>
</tr>
<tr>
<td>timeInMinutes</td>
<td>IccTime</td>
</tr>
<tr>
<td>timeInSeconds</td>
<td>IccTime</td>
</tr>
<tr>
<td>type</td>
<td>IccTime</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 14. IccAlarmRequestId class

IccBase
  IccResourceId
  IccRequestId
  IccAlarmRequestId

An IccAlarmRequestId object represents a unique alarm request. 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

Constructor (1)

IccAlarmRequestId()

Creates a new object with no information present.

Constructor (2)

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.

Creates an object with information already set.

Constructor (3)

IccAlarmRequestId(const IccAlarmRequestId& id)

id  A reference to an IccAlarmRequestId object.
The copy constructor.

Public methods

isExpired

Icc::Bool isExpired()

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

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

setTimerECA

void setTimerECA(const void* timerECA)

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

timerECA

const void* timerECA() const

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

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 15. IccBase class

IccBase

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

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

The opt parameter
Many methods have the same parameter, opt, which is described under the abendCode method in "abendCode" on page 67.

classType

ClassType classType() const

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 60.

className

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.

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.

customClassNum
IccBase

unsigned short customClassNum() const

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

operator delete

void operator delete(void* object)

object
  A pointer to an object that is to be destroyed.
Destroys an object in an orderly manner.

operator new

void* operator new(size_t size)

size
  The size of the object that is to be created, in bytes.
Creates a new object of given size. This operator enables the Foundation Classes to use CICS storage allocation (see initializeEnvironment on page 63).

Protected methods

setClassName

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").
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.

setCustomClassNum

void setCustomClassNum(unsigned short number)

number
  The number that an application designer associates with a subclass for identification purposes.
Assigns an identification number to a subclass that is not an original part of the classes, as supplied.

Enumerations

ClassType

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cAbendData</td>
<td>cGroupId</td>
<td>cSystem</td>
</tr>
</tbody>
</table>

### IccBase

<table>
<thead>
<tr>
<th>cAlarmRequestId</th>
<th>cJournal</th>
<th>cTask</th>
</tr>
</thead>
<tbody>
<tr>
<td>cBuf</td>
<td>cJournalId</td>
<td>cTempStore</td>
</tr>
<tr>
<td>cClock</td>
<td>cJournalTypeld</td>
<td>cTempStoreId</td>
</tr>
<tr>
<td>cConsole</td>
<td>cLockId</td>
<td>cTermId</td>
</tr>
<tr>
<td>cControl</td>
<td>cMessage</td>
<td>cTerminal</td>
</tr>
<tr>
<td>cConvId</td>
<td>cPartnerId</td>
<td>cTerminalData</td>
</tr>
<tr>
<td>cCUSTOM</td>
<td>cProgram</td>
<td>cTime</td>
</tr>
<tr>
<td>cDataQueue</td>
<td>cProgramId</td>
<td>cTPNameId</td>
</tr>
<tr>
<td>cDataQueueId</td>
<td>cRecordIndex</td>
<td>cTransId</td>
</tr>
<tr>
<td>cEvent</td>
<td>cRequestld</td>
<td>cUser</td>
</tr>
<tr>
<td>cException</td>
<td>cSemaphore</td>
<td>cUserId</td>
</tr>
<tr>
<td>cFile</td>
<td>cSession</td>
<td></td>
</tr>
<tr>
<td>cFileld</td>
<td>cStartRequestQ</td>
<td></td>
</tr>
<tr>
<td>cFileIterator</td>
<td>cSysId</td>
<td></td>
</tr>
</tbody>
</table>

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

### NameOpt

See ["className"](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**.
IccBase
Chapter 16. IccBuf class

IccBuf class is supplied for the general manipulation of buffers. This class is used by other classes that make calls to CICS, but does not itself call CICS services. See [Chapter 6. Buffer objects on page 23](#).

Header file: ICCBUFEH

Sample: ICC$BUF

IccBuf constructors

Constructor (1)

```
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.

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

Constructor (2)

```
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.

Creates an IccBuf object that cannot be extended, adopting the given data area as its own.

See warning about [Internal/External ownership of buffers on page 23](#).

Constructor (3)

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

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

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

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.

**Constructor (4)**

```
IccBuf(const IccBuf& buffer)
```

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

**Public methods**

**append (1)**

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

**append (2)**

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

*format*
The null-terminated format string
*...*The optional parameters.
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.

**assign (1)**

```
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.
Assigns data from the given data area to the data area in the object.
assign (2)

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

format
The format string
...
The optional parameters.
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.

cut

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

dataArea

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

offset
The offset into the data area. The default is no offset.
Returns the address of data at the given offset into the data area.

dataAreaLength

unsigned long dataAreaLength() const

Returns the length of the data area in bytes.

dataAreaOwner

DataAreaOwner dataAreaOwner() const

Returns an enumeration that indicates whether the data area has been allocated by the IccBuf constructor or has been supplied from elsewhere. The possible values are listed under "DataAreaOwner" on page 90.

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 90.
Returns the length of data in the data area. This cannot be greater than the value returned by `dataAreaLength`.

**insert**

```cpp
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.

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

**isFMHContained**

```cpp
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**

```cpp
operator const char*() const
```

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

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

**operator=(1)**

```cpp
IccBuf& operator=(const IccBuf& buffer)
```

- `buffer`: A reference to an `IccBuf` object.

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

**operator=(2)**

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

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

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

**operator+= (1)**

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

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

**operator+= (2)**

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

*text*  
The null-terminated string to be appended to the *IccBuf* object.  
Appends data from a null-terminated string and returns a reference to the *IccBuf* object.

See also the `append` method.

**operator==**

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

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

**operator!=**

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

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

**operator<<(1)**

```
operator<<(const IccBuf& buffer)
```

Appends another buffer.

**operator<<(2)**

```
operator<<(const char* text)
```

Appends a string.

**operator<<(3)**

```
operator<<(char ch)
```

Appends a character.
IccBuf

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

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

length
   The length of the existing data area.
dataArea
   The address of the existing data area.
Makes the data area external and fixed. Any existing internal data area is destroyed.

See warning about “Internal/External ownership of buffers” on page 23.

replace

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

setDataLength

unsigned long setDataLength(unsigned long length)

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

setFMHContained

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

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

Chapter 16. IccBuf class 89
**IccBuf**

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

**Inherited protected methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

**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.
Chapter 17. IccClock class

IccBase
   IccResource
      IccClock

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

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

absTime

IccAbsTime& absTime()

Returns a reference to an IccAbsTime object that contains the absolute time as
provided by CICS.

cancelAlarm

void cancelAlarm(const IccRequestId* reqId = 0)

reqId
   An optional pointer to the IccRequestId object that holds information on an
   alarm request.
   Cancels a previous setAlarm request if the alarm time has not yet been reached,
   that is, the request has not expired.

   Conditions
   ISCINVREQ, NOTAUTH, NOTFND, SYSIDERR

date

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

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.
Returns the date as a string.

Conditions
   INVREQ

dayOfMonth

unsigned long dayOfMonth()

Returns the day component of the date, in the range 1 to 31.

Conditions
   INVREQ

dayOfWeek

DayOfWeek dayOfWeek()

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

Conditions
   INVREQ

daysSince1900

unsigned long daysSince1900()

Returns the number of days that have elapsed since 1st January, 1900.

Conditions
   INVREQ

milliSeconds

long double milliSeconds()

Returns the number of milliseconds, rounded to the nearest hundredth of a second,
that have elapsed since 00:00 on 1st January, 1900.

monthOfYear

MonthOfYear monthOfYear()

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

Conditions
   INVREQ
**setAlarm**

```cpp
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 practice, 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.

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 method on page 213 of class IccTask.

**Conditions**

EXPIRED, INVREQ

**time**

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

*timeSeparator*

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

Returns the time as a text string.

**Conditions**

INVREQ

**update**

```cpp
void update()
```

Updates the clock time and date from CICS. See the IccClock constructor.

**year**

```cpp
unsigned long year()
```

Returns the 4-figure year number, such as 1996.

**Conditions**

INVREQ

**Inherited public methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
</tbody>
</table>
## IccClock

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

### Enumerations

#### DateFormat

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultFormat</td>
<td></td>
</tr>
<tr>
<td>DDMMYY</td>
<td></td>
</tr>
<tr>
<td>MMDDYY</td>
<td></td>
</tr>
<tr>
<td>YYDDD</td>
<td></td>
</tr>
<tr>
<td>YYDDMM</td>
<td></td>
</tr>
<tr>
<td>YYMMDD</td>
<td></td>
</tr>
<tr>
<td>DDMMYYYY</td>
<td></td>
</tr>
<tr>
<td>MMDDYYYY</td>
<td></td>
</tr>
<tr>
<td>YYYYDDDD</td>
<td></td>
</tr>
<tr>
<td>YYYYDDMM</td>
<td></td>
</tr>
<tr>
<td>YYYYMMDD</td>
<td></td>
</tr>
</tbody>
</table>

#### DayOfWeek

Indicates the day of the week.
- Sunday
- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday

#### MonthOfYear

Indicates the month of the year.
- January
- February
- March
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`).
IccClock
Chapter 18. IccCondition structure

This structure contains an enumeration of all the CICS condition codes.

Header file: ICCCNDEH

Enumerations

Codes

The possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NORMAL</td>
<td>35</td>
</tr>
<tr>
<td>1</td>
<td>ERROR</td>
<td>36</td>
</tr>
<tr>
<td>2</td>
<td>RDATT</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>WBRK</td>
<td>38</td>
</tr>
<tr>
<td>4</td>
<td>ICCEOF</td>
<td>39</td>
</tr>
<tr>
<td>5</td>
<td>EODS</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>EOC</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>INBFMH</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>ENDINPT</td>
<td>43</td>
</tr>
<tr>
<td>9</td>
<td>NONVAL</td>
<td>44</td>
</tr>
<tr>
<td>10</td>
<td>NOSTART</td>
<td>45</td>
</tr>
<tr>
<td>11</td>
<td>TERMIDERR</td>
<td>46</td>
</tr>
<tr>
<td>12</td>
<td>FILENOTFOUND</td>
<td>47</td>
</tr>
<tr>
<td>13</td>
<td>NOTFND</td>
<td>48</td>
</tr>
<tr>
<td>14</td>
<td>DUPREC</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>DUPKEY</td>
<td>50</td>
</tr>
<tr>
<td>16</td>
<td>INVREQ</td>
<td>51</td>
</tr>
<tr>
<td>17</td>
<td>IOERR</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>NOSPACE</td>
<td>53</td>
</tr>
<tr>
<td>19</td>
<td>NOTOPEN</td>
<td>54</td>
</tr>
<tr>
<td>20</td>
<td>ENDFILE</td>
<td>55</td>
</tr>
<tr>
<td>21</td>
<td>ILLOGIC</td>
<td>56</td>
</tr>
<tr>
<td>22</td>
<td>LENGERR</td>
<td>57</td>
</tr>
<tr>
<td>23</td>
<td>QZERO</td>
<td>58</td>
</tr>
<tr>
<td>24</td>
<td>SIGNAL</td>
<td>59</td>
</tr>
<tr>
<td>25</td>
<td>QBUSY</td>
<td>60</td>
</tr>
<tr>
<td>26</td>
<td>ITEMERR</td>
<td>61</td>
</tr>
<tr>
<td>27</td>
<td>PGMIDERR</td>
<td>62</td>
</tr>
<tr>
<td>28</td>
<td>TRANSIDERR</td>
<td>63</td>
</tr>
<tr>
<td>29</td>
<td>ENDDATA</td>
<td>64</td>
</tr>
<tr>
<td>30</td>
<td>INVTSEQ</td>
<td>65</td>
</tr>
<tr>
<td>31</td>
<td>EXPIRED</td>
<td>66</td>
</tr>
<tr>
<td>32</td>
<td>RETPAGE</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>RETFAIL</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>RTESOME</td>
<td>69</td>
</tr>
</tbody>
</table>

Range

maxValue

The highest CICS condition, currently 103.
IccCondition
Chapter 19. IccConsole class

IccBase
   IccResource
      IccConsole

This is a singleton class that represents the CICS console.

Header file: ICCONEH

Sample: ICC$CON

IccConsole constructor (protected)

Constructor

IccConsole()

No more than one of these objects is permitted in a task. An attempt to create more objects causes an exception to be thrown.

Public methods

The opt parameter

Many methods have the same parameter, opt, which is described under the abendCode method in "abendCode" on page 67.

instance

static IccConsole* 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.

put

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.

Writes the data in send to the CICS console. put is a synonym for write. See "Polymorphic Behavior" on page 55.

replyTimeout

unsigned long replyTimeout() const

Returns the length of the reply timeout in milliseconds.
IccConsole

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)

    void setReplyTimeout(unsigned long seconds)

    seconds
        The length of the time interval required, in seconds.

The two different forms of this method are used to set the length of the reply
timeout.

setRouteCodes

    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.

Saves route codes in the object for use on subsequent write and
writeAndGetReply calls. Up to 28 codes can be held in this way.

write

    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, defined below, that indicates the severity of the console
        message.

Writes the data in send to the CICS console.
writeAndGetReply

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, defined below, that indicates the severity of the console
   message.

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.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFO</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Enumerations

SeverityOpt

Possible values are:
none
warning
error
severe
Chapter 20. IccControl class

IccBase
   IccResource
   IccControl

IccControl class controls an application program that uses the supplied Foundation Classes. 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

callingProgramId

const IccProgramId& 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.

Conditions
   INVREQ

cancelAbendHandler

void cancelAbendHandler()

Cancels a previously established exit at this logical program level.

Conditions
   NOTAUTH, PGMIDERR

commArea

IccBuf& 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.
IccControl

Conditions
INVREQ

console

IccConsole* 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.

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

static IccControl* instance()

Returns a pointer to the single IccControl object. The object is created if it does not already exist.

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

void resetAbendHandler()

Reactivates a previously cancelled abend handler for this logical program level. (See cancelAbendHandler on page 103).

Conditions
NOTAUTH, PGMIDERR
returnProgramId

    const IccProgramId& returnProgramId()

    Returns a reference to an IccProgramId object that refers to the program that
resumes control when this logical program level issues a return.

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)

    void setAbendHandler(const char* programName)

    programName
        The name of the program affected.
These methods set the abend handler to the named program for this logical
program level.

    Conditions
        NOTAUTH, PGMIDERR

startRequestQ

    IccStartRequestQ* 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.

system

    IccSystem* system()
IccControl

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 principle facility. That is, there must be a physical terminal involved.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccBase</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 21. IccConvId class

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)

IccConvId(const IccConvId& convId)

convId
   A reference to an IccConvId object.
   The copy constructor.

Public methods

operator= (1)

IccConvId& operator=(const char* convName)

operator= (2)

IccConvId& operator=(const IccConvId id)

Assigns new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccBase</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setclassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 22. IccDataQueue class

IccBase
  IccResource
  IccDataQueue

This class represents a CICS transient data queue.

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

clear

  virtual void clear()

    A synonym for empty. See "Polymorphic Behavior" on page 55.

empty

  void empty()

    Empties the queue, that is, deletes all items on the queue.

    Conditions
      ISCINVREQ, NOTAUTH, QIDERR, SYSIDERR, DISABLED, INVREQ

get

  virtual const IccBuf& get()

    A synonym for readItem. See "Polymorphic Behavior" on page 55.
### put

**virtual void put(const IccBuf& buffer)**

*buffer*  
A reference to an **IccBuf** object that contains data to be put into the queue.  
A synonym for **writeItem**. See [Polymorphic Behavior](#) on page 55.

### 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)

**void writeItem(const char* text)**

*text*  
Text that is to be written to the queue.  
Writes an item of data to the queue.

**Conditions**  
IOERR, ISCINVREQ, LENGERR, NOSPACE, NOTAUTH, NOTOPEN, QIDERR, SYSIDERR, DISABLED, INVREQ

### Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>isRouteOptionOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
### IccDataQueue

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>routeOption</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
<tr>
<td>setRouteOption</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
IccDataQueue
Chapter 23. IccDataQueueId class

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

operator= (1)

IccDataQueueId& operator=(const char* queueName)

queueName
   The 4-character name of the queue

operator= (2)

IccDataQueueId& operator=(const IccDataQueueId& id)

id  A reference to an IccDataQueueId object. Assigns new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 24. IccEvent class

IccBase
    IccEvent

The IccEvent class contains information on a particular CICS call, which we call a CICS event.

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

className

const char* className() const

Returns the name of the class responsible for this event.

classType

IccBase::ClassType classType() const

Returns an enumeration, described under classType on page 79 in IccBase class, that indicates the type of class that is responsible for this event.

condition

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.

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.
IccEvent

conditionText

```cpp
const char* conditionText() const
```

Returns the text of the CICS condition code, such as "NORMAL" or "LENGERR".

methodName

```cpp
const char* methodName() const
```

Returns the name of the method responsible for this event.

summary

```cpp
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

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>lccBase</td>
</tr>
<tr>
<td>classType</td>
<td>lccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>lccBase</td>
</tr>
<tr>
<td>operator delete</td>
<td>lccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>lccBase</td>
</tr>
</tbody>
</table>

---

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setclassName</td>
<td>lccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>lccBase</td>
</tr>
</tbody>
</table>
Chapter 25. IccException class

IccBase
IccException

IccException class contains information about CICS Foundation Class exceptions. 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

className

const char* className() const

Returns the name of the class responsible for throwing this exception.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

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. For example, an attempt to create an IccJournal object would fail under CICS for OS/2 because there are no CICS journal services on this server.

A platformError exception can occur at 3 levels:
1. An object is not supported on this platform.
**IccException**

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 on page 52](#) 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.
Chapter 26. IccFile class

IccBase
   IccResource
      IccFile

IccFile class enables the application program to access CICS files.

Header file: ICCFILEH

Sample: ICC$FIL

IccFile constructors

Constructor (1)

\[
\text{IccFile (const IccFileId& } \ id, \\
\text{IccRecordIndex* index } = 0)
\]

\text{id} A reference to the \text{IccFileId} object that identifies which file is being operated on
\text{index} An optional pointer to the \text{IccRecordIndex} object that identifies which record in

the file is being operated on.

Constructor (2)

\[
\text{IccFile (const char* } \ fileName, \\
\text{IccRecordIndex* index } = 0)
\]

\text{fileName} The 8-character name of the file
\text{index} An optional pointer to the \text{IccRecordIndex} object that identifies which record in

the file is being operated on.

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.

Public methods

<table>
<thead>
<tr>
<th>The opt parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many methods have the same parameter, \text{opt}, which is described under the \text{abendCode} method in \text{abendCode} on page 67</td>
</tr>
</tbody>
</table>

access

\[
\text{unsigned long access(Icc::GetOpt } \ opt \ =\text{Icc::object)}
\]

\text{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.
IccFile

Returns a composite number indicating the access properties of the file. See also isReadable, isBrowsable, isAddable, isDeletable, and isUpdatable methods.

accessMethod

IccValue::CVDA accessMethod(Icc::GetOpt opt = Icc::object)

opt

See access method.

Returns an enumeration, defined in IccValue, that represents the access method for this file. Possible values are:

- VSAM
- BDAM
- SFS

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

beginInsert(VSAM only)

void beginInsert()

Signals the start of a mass insertion of data into the file.

deleteLockedRecord

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.

Deletes a record that has been previously locked by readRecord method in update mode. (See also readRecord method.)

Conditions

DISABLED, DUPKEY, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFIND, NOTOPEN, SYSIDERR, LOADING

deleteRecord

unsigned short deleteRecord()

Deletes one or more records, as specified by the associated IccRecordIndex object, and returns the number of deleted records.

Conditions

DISABLED, DUPKEY, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFIND, NOTOPEN, SYSIDERR, LOADING

enableStatus

IccValue::CVDA enableStatus(Icc::GetOpt opt = Icc::object)

opt

See access method.
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

**Conditions**

| END | FILENOTFOUND | ILLOGIC | NOTAUTH |

**endInsert (VSAM only)**

```cpp
void endInsert()
```

Marks the end of a mass insertion operation. See **beginInsert**.

**isAddable**

```cpp
Icc::Bool isAddable(Icc::GetOpt opt = Icc::object)
```

*opt*

- See **access** method.
- Indicates whether more records can be added to the file.

**Conditions**

| END | FILENOTFOUND | ILLOGIC | NOTAUTH |

**isBrowsable**

```cpp
Icc::Bool isBrowsable(Icc::GetOpt opt = Icc::object)
```

*opt*

- See **access** method.
- Indicates whether the file can be browsed.

**Conditions**

| END | FILENOTFOUND | ILLOGIC | NOTAUTH |

**isDeletable**

```cpp
Icc::Bool isDeletable(Icc::GetOpt opt = Icc::object)
```

*opt*

- See **access** method.
- Indicates whether the records in the file can be deleted.

**Conditions**

| END | FILENOTFOUND | ILLOGIC | NOTAUTH |

**isEmptyOnOpen**

```cpp
Icc::Bool isEmptyOnOpen(Icc::GetOpt opt = Icc::object)
```

*opt*

- See **access** method.
**IccFile**

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.

**Conditions**
END, FILENOTFOUND, ILLOGIC, NOTAUTH

**isReadable**

```cpp
Icc::Bool isReadable(Icc::GetOpt opt = Icc::object)
```

*opt*
See `access` method.
Indicates whether the file records can be read.

**Conditions**
END, FILENOTFOUND, ILLOGIC, NOTAUTH

**isRecoverable**

```cpp
Icc::Bool isRecoverable(Icc::GetOpt opt = Icc::object)
```

*opt*
See `access` method.

**Conditions**: END, FILENOTFOUND, ILLOGIC, NOTAUTH

**isUpdatable**

```cpp
Icc::Bool isUpdatable(Icc::GetOpt opt = Icc::object)
```

*opt*
See `access` method.
Indicates whether the file can be updated.

**Conditions**
END, FILENOTFOUND, ILLOGIC, NOTAUTH

**keyLength**

```cpp
unsigned long keyLength(Icc::GetOpt opt = Icc::object)
```

*opt*
See `access` method.
Returns the length of the search key.

**Conditions**
END, FILENOTFOUND, ILLOGIC, NOTAUTH

**keyPosition**

```cpp
long keyPosition(Icc::GetOpt opt = Icc::object)
```

*opt*
See `access` method.
IccFile

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.

**Conditions**
END, FILENOTFOUND, ILLOGIC, NOTAUTH

**openStatus**

IccValue::CVDA openStatus(Icc::GetOpt opt = Icc::object)

*opt*
See access method.

Returns a CVDA that indicates the open status of the file. Possible values are:

**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

**recordFormat**

IccValue::CVDA recordFormat(Icc::GetOpt opt = Icc::object)

*opt*
See access method.

Returns a CVDA that indicates the format of the data. Possible values are:

**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**

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 wish to make multiple read updates. The token uniquely identifies the update request and is passed to the deleteLockedRecord, rewriteRecord, or unlockRecord methods.

Reads a record and returns a reference to an IccBuf object that contains the data from the record.

**Conditions**
DISABLED, DUPKEY, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR, LOADING

Chapter 26. IccFile class 125
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**

```cpp
IccRecordIndex* recordIndex() const
```

Returns a pointer to an `IccRecordIndex` object that indicates which records are to be accessed when using methods such as `readRecord`, `writeRecord`, and `deleteRecord`.

**recordLength**

```cpp
unsigned long recordLength(Icc::GetOpt opt = Icc::object)
```

**opt**

See `access` method.

Returns the length of the current record.

**Conditions**

END, FILENOTFOUND, ILLOGIC, NOTAUTH

**registerRecordIndex**

```cpp
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**

```cpp
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`.

Updates a record with the contents of `buffer`.

**Conditions**

DISABLED, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR, LOADING
**setAccess**

```c
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.

Sets the permitted access to the file. For example:

```c
file.setAccess(IccFile::readable + IccFile::notUpdatable);
```

**Conditions**
FILENOTFOUND, INVREQ, IOERR, NOTAUTH

**setEmptyOnOpen**

```c
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**

```c
void setStatus(Status status)
```

**status**
An enumeration, defined in this class, that indicates the required status of the file after this method is called.

Sets the status of the file.

**Conditions**
FILENOTFOUND, INVREQ, IOERR, NOTAUTH

**type**

```c
IccValue::CVDA type(Icc::GetOpt opt = Icc::object)
```

**opt**
See *access* method.

Returns a CVDA that identifies the type of data set that corresponds to this file.

Possible values are:

- **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
Unlock a record, previously locked by reading it in update mode. See `readRecord`.

**Conditions**
DISABLED, FILENOTFOUND, ILLOGIC, IOERR, ISCINVREQ, NOTAUTH, NOTOPEN, SYSIDERR, INVREQ

Write either a single record or a sequence of records, if used with the `beginInsert` and `endInsert` methods.

**Conditions**
DISABLED, DUPREC, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOSPACE, NOTAUTH, NOTOPEN, SYSIDERR, LOADING, SUPPRESSED

### Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>isRouteOptionOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>routeOption</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
<tr>
<td>setRouteOption</td>
<td>IccResource</td>
</tr>
</tbody>
</table>
Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

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

- **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.
IccFile

disabled

File is disabled from access by CICS tasks.
Chapter 27. IccFileId class

IccFileId is used to identify a file name in the CICS system. On MVS/ESA this is an entry in the FCT (file control table).

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

operator= (1)

IccFileId& operator=(const char* fileName)

fileName
The 8-byte name of the file.

operator= (2)

IccFileId& operator=(const IccFileId& id)

id A reference to an IccFileId object. Assigns new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceld</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceld</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 28. IccFileIterator class

IccBase
   IccResource
         IccFileIterator

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.

Header file: ICCFLIEH

Sample: ICC$FIL

IccFileIterator constructor

Constructor

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.

The IccFile and IccRecordIndex object must exist before the IccFileIterator is created.

Conditions
DISABLED, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR, LOADING

Public methods

readNextRecord

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.

Read the record that follows the current record.

Conditions
DUPKEY, ENDFILE, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFND, SYSIDERR
readPreviousRecord

```
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.
Read the record that precedes the current record.

**Conditions**
DUPKEY, ENDFILE, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFIND, SYSIDERR

reset

```
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.

Resets the IccFileIterator object to point to the record identified by the IccRecordIndex object and the specified search criterion.

**Conditions**
FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFND, SYSIDERR

**Inherited public methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>isRouteOptionOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>routeOption</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
</tbody>
</table>
### Method Class

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
<tr>
<td>setRouteOption</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 29. IccGroupId class

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)

IccGroupId(const IccGroupId& id)

id A reference to an IccGroupId object.
The copy constructor.

Public methods

operator= (1)

IccGroupId& operator=(const char* groupName)

groupName
The 8-character name of the group.

operator= (2)

IccGroupId& operator=(const IccGroupId& id)

id A reference to an IccGroupId object.
Assigns new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
IccGroupId

## Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 30. IccJournal class

IccBase
   IccResource
      IccJournal

IccJournal class represents a user or system CICS journal.

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

clearPrefix

void 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.
IccJournal

**journalTypeld**

```cpp
const IccJournalTypeld& journalTypeld() const
```

Returns a reference to an `IccJournalTypeld` object that contains a 2-byte field used to identify the origin of journal records.

**put**

```cpp
virtual void put(const IccBuf& buffer)
```

*buffer*

A reference to an `IccBuf` object that holds data to be put into the journal. A synonym for `writeRecord`—puts data into the journal. See [Polymorphic Behavior](#) on page 55 for information on polymorphism.

**registerPrefix**

```cpp
void registerPrefix(const IccBuf* prefix)
```

Stores pointer to prefix object for use when the `writeRecord` method is called on this `IccJournal` object.

**setJournalTypeld (1)**

```cpp
void setJournalTypeld(const IccJournalTypeld& id)
```

**setJournalTypeld (2)**

Sets the journal type—a 2 byte identifier—included in the journal record created when using the `writeRecord` method.

```cpp
void setJournalTypeld(const char* jtypeid)
```

**setPrefix (1)**

```cpp
void setPrefix(const IccBuf& prefix)
```

**setPrefix (2)**

Stores the current contents of `prefix` for inclusion in the journal record created when the `writeRecord` method is called.

```cpp
void setPrefix(const char* prefix)
```

**wait**

```cpp
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 \texttt{IccJournal::start10 + IccJournal::synchronous}. The
default is to use the system default.

Waits until a previous journal write has completed.

\textbf{Condition:} IOERR, JIDERR, NOTOPEN

\textbf{writeRecord (1)}

\begin{verbatim}
unsigned long writeRecord (const IccBuf& record, 
unsigned long option = 0)

record
  A reference to an \texttt{IccBuf} object that holds the record

option
  See above.
\end{verbatim}

\textbf{writeRecord (2)}

\begin{verbatim}
unsigned long writeRecord (const char* record, 
unsigned long option = 0)

record
  The name of the record

option
  See above.

Writes the data in the record to the journal.

The returned number represents the particular write request and can be passed to the \texttt{wait} method in this class.

\textbf{Conditions}

IOERR, JIDERR, LENGERR, NOJBUFSP, NOTAUTH, NOTOPEN

\underline{Inherited public methods}

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>classType</td>
<td>\texttt{IccBase}</td>
</tr>
<tr>
<td>className</td>
<td>\texttt{IccBase}</td>
</tr>
<tr>
<td>condition</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>conditionText</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>customClassNum</td>
<td>\texttt{IccBase}</td>
</tr>
<tr>
<td>handleEvent</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>id</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>name</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>operator delete</td>
<td>\texttt{IccBase}</td>
</tr>
<tr>
<td>operator new</td>
<td>\texttt{IccBase}</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>\texttt{IccResource}</td>
</tr>
<tr>
<td>setEDF</td>
<td>\texttt{IccResource}</td>
</tr>
</tbody>
</table>
Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

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.
Chapter 31. IccJournalId class

IccBase
   IccResourceId
   IccJournalId

IccJournalId is used to identify a journal number in the CICS system. On MVS/ESA this is an entry in the JCT (Journal Control Table).

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)

IccJournalId(const IccJournalId& id)

id   A reference to an IccJournalId object.
     The copy constructor.

Public methods

number

unsigned short number() const

Returns the journal number, in the range 1 to 99.

operator= (1)

IccJournalId& operator=(unsigned short journalNum)

journalNum
   The number of the journal, in the range 1 to 99

operator= (2)

IccJournalId& operator=(const IccJournalId& id)

id   A reference to an IccJournalId object.
     Assigns new value.
### Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 32. IccJournalTypeId class

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: ICCRIDEM

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

operator= (1)

void operator=(const IccJournalTypeId& id)

id A reference to an IccJournalTypeId object.

operator= (2)

void operator=(const char* journalTypeName)

journalTypeName
A 2-byte identifier used in journal records.
Sets the 2-byte field that is included in the journal record.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 33. IccKey class

IccBase
  IccRecordIndex
     IccKey

IccKey class is used to hold a search key for an indexed (KSDS) file.

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

assign

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.
Copies the search key into the IccKey object.

completeLength

unsigned short completeLength() const

Returns the length of the key when it is complete.

kind

Kind kind() const
IccKey

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

void setKind(Kind kind)

kind

An enumeration, defined in this class, that indicates whether the key is generic or complete.
Changes the type of key from generic to complete or vice versa.

value

const char* value()

Returns the start of the data area containing the search key.
Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>length</td>
<td>IccRecordIndex</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>type</td>
<td>IccRecordIndex</td>
</tr>
<tr>
<td>value</td>
<td>IccRecordIndex</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

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.
IccKey
Chapter 34. IccLockId class

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)

IccLockId(const IccLockId& id)

id   A reference to an IccLockId object.
     The copy constructor.

Public methods

operator= (1)

IccLockId& operator=(const char* name)

name
   The 8-character name of the lock request.

operator= (2)

IccLockId& operator=(const IccLockId& id)

id   A reference to an IccLockId object.
     Assigns new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 35. IccMessage class

IccBase
   IccMessage

IccMessage can be used to hold a message description. 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

className

const char* className() const

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.

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
IccMessage

    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

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 36. IccPartnerId class

IccBase
    IccResourceld
    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)

IccPartnerId(const IccPartnerId& id)

id A reference to an IccPartnerId object.
The copy constructor.

Public methods

operator=(1)

IccPartnerId& operator=(const char* partnerName)

partnerName
The 8-character name of an APPC partner.

operator=(2)

IccPartnerId& operator=(const IccPartnerId& id)

id A reference to an IccPartnerId object.
Assigns new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceld</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceld</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
## Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operator=</code></td>
<td><code>IccResourceId</code></td>
</tr>
<tr>
<td><code>setClassName</code></td>
<td><code>IccBase</code></td>
</tr>
<tr>
<td><code>setCustomClassNum</code></td>
<td><code>IccBase</code></td>
</tr>
</tbody>
</table>

**IccPartnerId**
Chapter 37. IccProgram class

IccBase
   IccResource
   IccProgram

The IccProgram class represents any CICS program outside of your currently executing one, which the IccControl object represents.

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 section "abendCode" on page 67.

address

const void* address() const

Returns the address of a program module in memory. This is only valid after a successful load call.

clearInputMessage

void 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.

**entryPoint**

```cpp
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**

```cpp
unsigned long length() const
```

Returns the length of a program module. This is only valid after a successful `load` call.

**link**

```cpp
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**

```cpp
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).
registerInputMessage

```cpp
void registerInputMessage(const IccBuf& msg)
```

Store pointer to InputMessage for when the `link` method is called.

setInputMessage

Specifies data to be made available, by the `IccSession::receive()` method, to the `void setInputMessage(const IccBuf& msg)` called program, when using the `link` method in this class.

unload

```cpp
void unload()
```

Allow a program to be unloaded. It can be reloaded by a call to `load`.

**Conditions**

NOTAUTH, PGMIDERR, INVREQ

---

### Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>actionOnCondition</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>actionOnConditionAsChar</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>actionsOnConditionsText</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>className</code></td>
<td><code>IccBase</code></td>
</tr>
<tr>
<td><code>classType</code></td>
<td><code>IccBase</code></td>
</tr>
<tr>
<td><code>condition</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>conditionText</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>customClassNum</code></td>
<td><code>IccBase</code></td>
</tr>
<tr>
<td><code>handleEvent</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>id</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>isEDFOn</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>isRouteOptionOn</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>name</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>operator delete</code></td>
<td><code>IccBase</code></td>
</tr>
<tr>
<td><code>operator new</code></td>
<td><code>IccBase</code></td>
</tr>
<tr>
<td><code>routeOption</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>setActionOnAnyCondition</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>setActionOnCondition</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>setActionsOnConditions</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>setEDF</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>setRouteOption</code></td>
<td><code>IccResource</code></td>
</tr>
</tbody>
</table>

---

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setName</code></td>
<td><code>IccResource</code></td>
</tr>
<tr>
<td><code>setCustomClassNum</code></td>
<td><code>IccResource</code></td>
</tr>
</tbody>
</table>

---

Chapter 37. IccProgram class  159
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).
Chapter 38. IccProgramId class

IccBase
   IccResourceId
      IccProgramId

IccProgramId objects represent program names in the CICS system. On MVS/ESA this is an entry in the PPT (program processing table).

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)

IccProgramId& operator=(const IccProgramId& id)

id   A reference to an IccProgramId object. Assigns new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>Method</td>
<td>Class</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 39. IccRBA class

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)

IccRBA& operator=(unsigned long num)

num
    A valid relative byte address.
    Assigns a new value for the 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
## IccRBA

### number

**unsigned long number() const**

Returns the relative byte address.

## Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>length</td>
<td>IccRecordIndex</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>type</td>
<td>IccRecordIndex</td>
</tr>
<tr>
<td>value</td>
<td>IccRecordIndex</td>
</tr>
</tbody>
</table>

## Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 40. IccRecordIndex class

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 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

unsigned short length() const

Returns the length of the record identifier.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

### Enumerations

**Type**

Indicates the access method. Possible values are:
- key
- RBA
- RRN
Chapter 41. IccRequestId class

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)

IccRequestId()

An empty IccRequestId object.

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)

IccRequestId& operator=(const char* requestName)

requestName
   An 8-character string which is copied into this object.
   Assigns new value.
# IccRequestId

## Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

## Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 42. IccResource class

IccBase

IccResource

IccResource class is a base class that is used to derive other classes. 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

ActionOnCondition actionOnCondition(IccCondition::Codes condition)

condition
The name of the condition as an enumeration. See IccCondition structure for a list of the possible values.

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.

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

const char* 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 above in "actionOnConditionAsChar" on page 168. 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)

clear

virtual void clear()

Clears the contents of the object. This method is virtual and is implemented, wherever appropriate, in the derived classes. See "Polymorphic Behavior" on page 55 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.

condition

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. Returns a number that indicates the condition code for the most recent CICS call made by this object.

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 \texttt{IccResource} object and returns it as an \texttt{IccBuf} reference. This method is virtual and is implemented, wherever appropriate, in the derived classes. See \cite{Polymorphic Behavior on page 55} 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.

\subsubsection*{handleEvent}

\begin{verbatim}
virtual HandleEventReturnOpt handleEvent(IccEvent& event)
\end{verbatim}

\textit{event}
\begin{description}
\item A reference to an \texttt{IccEvent} object that describes the reason why this method is being called.
\end{description}

This virtual function may be re-implemented in a subclass (by the application programmer) to handle CICS events (see \texttt{IccEvent} class on page \texttt{115}).

\subsubsection*{id}

\begin{verbatim}
const IccResourceId* id() const
\end{verbatim}

Returns a pointer to the \texttt{IccResourceId} object associated with this \texttt{IccResource} object.

\subsubsection*{isEDFOn}

\begin{verbatim}
Icc::Bool isEDFOn() const
\end{verbatim}

Returns a boolean value that indicates whether EDF trace is active. Possible values are yes or no.

\subsubsection*{isRouteOptionOn}

\begin{verbatim}
Icc::Bool isRouteOptionOn() const
\end{verbatim}

Returns a boolean value that indicates whether the route option is active. Possible values are yes or no.

\subsubsection*{name}

\begin{verbatim}
const char* name() const
\end{verbatim}

Returns a character string that gives the name of the resource that is being used. For an \texttt{IccTempStore} object, the 8-character name of the temporary storage queue is returned. For an \texttt{IccTerminal} object, the 4-character terminal name is returned. This is equivalent to calling \texttt{id()}\texttt{.name}.

\subsubsection*{put}

\begin{verbatim}
virtual void put(const IccBuf& buffer)
\end{verbatim}

\textit{buffer}
\begin{description}
\item A reference to an \texttt{IccBuf} object that contains data that is to be put into the object.
\end{description}
IccResource

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" on page 55 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.

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

void setActionOnAnyCondition(ActionOnCondition action)

action
The name of the action as an enumeration. The possible values are listed under the description of this class.
Specifies the default action to be taken by the CICS foundation classes when a CICS condition occurs.

setActionOnCondition

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.
Specifies what action is automatically taken by the CICS foundation classes when a given CICS condition occurs.

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

void setEDF(Icc::Bool onOff)

onOff
A boolean value that selects whether EDF trace is switched on or off.
Switches EDF on or off for this resource object. See "Execution Diagnostic Facility" on page 46.
These methods force the object to route CICS requests to the named remote system. This is called explicit function shipping.

**setRouteOption (1)**

The parameters are:

```c
void setRouteOption(const IccSysId& sysId)
```

*sysId*

The **IccSysId** object that represents the remote system to which commands are routed.

**setRouteOption (2)**

```c
void setRouteOption(const char* sysName = 0)
```

*sysName*

The 4-character name of the system to which commands are routed.

This option is only valid for the following classes:

- **IccDataQueue**
- **IccFile**
- **IccFileIterator**
- **IccProgram**
- **IccStartRequestQ**
- **IccTempStore**

Attempting to use this method on other subclasses of **IccResource** causes an exception to be thrown.

To turn off the route option specify no parameter, for example:

```c
obj.setRouteOption()
```

### Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

### Enumerations

**ActionOnCondition**

Possible values are:
IccResource

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.
Chapter 43. IccResourceId class

IccBase
   IccResourceId

This is a base class from which IccTransId and other classes, whose names all end in "Id", are derived. Many of these derived classes represent CICS resource names, such as a file control table (FCT) entry.

Header file: ICCRIDEH

IccResourceId constructors (protected)

Constructor (1)

IccResourceId (IccBase::ClassType type,
   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

name

   const char* name() const

Returns the name of the resource identifier as a string. Most ...Id objects have 4- or 8-character names.

nameLength

   unsigned short nameLength() const

Returns the length of the name returned by the name method.
IccResourceld

Protected methods

operator=

IccResourceld& operator=(const IccResourceld& id)

id A reference to an IccResourceld object.
Set an IccResourceld object to be identical to id.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 44. IccRRN class

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

operator=(1)

IccRRN& operator=(const IccRRN& rrn)

operator=(2)

IccRRN& operator=(unsigned long num)

num
A relative record number—an integer greater than 0. Assigns a new value for the relative record number.

operator==(1)

Icc::Bool operator==(const IccRRN& rrn) const

operator==(2)

Icc::Bool operator==(unsigned long num) const

Tests equality

operator!=(1)

Icc::Bool operator!=(const IccRRN& rrn) const

operator!=(2)

Icc::Bool operator!=(unsigned long num) const
Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>length</td>
<td>IccRecordIndex</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>type</td>
<td>IccRecordIndex</td>
</tr>
<tr>
<td>value</td>
<td>IccRecordIndex</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 45. IccSemaphore class

IccBase
    IccResource
        IccSemaphore

This class enables synchronization of resource updates.

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

lifeTime

LifeTime lifeTime() const

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').
**IccSemaphore**

**lock**

```cpp
void lock()
```

Attempts to get a lock. This method blocks if another task already owns the lock.

**Conditions**
ENQBUSY, LENGERR, INVREQ

**tryLock**

```cpp
Icc::Bool 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.

**Conditions**
ENQBUSY, LENGERR, INVREQ

**type**

```cpp
LockType type() const
```

Returns an enumeration, defined in this class, that indicates what type of semaphore this is.

**unlock**

```cpp
void unlock()
```

Release a lock.

**Conditions**
LENGERR, INVREQ

---

**Inherited public methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
</tbody>
</table>
Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

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.
IccSemaphore
Chapter 46. IccSession class

```
IccBase
    IccResource
        IccSession
```

This class enables APPC and DTP programming.

**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**

```
IccSession()
```

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.

© Copyright IBM Corp. 1989, 2001
IccSession

Public methods

allocate

```cpp
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. Establishes a session (communication channel) to the remote system.

**Conditions**

INVREQ, SYSIDERR, CBIDERR, NETNAMEIDERR, PARTNERIDERR, SYSBUSY

connectProcess (1)

```cpp
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

This method can only be used if an IccPartnerId object was used to construct this session object.

connectProcess (2)

```cpp
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)

```cpp
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
Starts a partner process on the remote system in preparation for sending and receiving information.

Conditions
INVREQ, LENGERR, NOTALLOC, PARTNERIDERR, NOTAUTH, TERMERR, SYSBUSY

**converse**

```cpp
const IccBuf& converse(const IccBuf& send)
```

**send**
A reference to an IccBuf object that contains the data that is to be sent.
`converse` sends the contents of `send` and returns a reference to an IccBuf object that holds the reply from the remote APPC partner.

**Conditions**
EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

**convId**

```cpp
const IccConvId& convId()
```

Returns a reference to an IccConvId object that contains the 4-byte conversation identifier.

**errorCode**

```cpp
const char* errorCode() const
```

Returns the 4-byte error code received when `isErrorSet` returns true. See the relevant DTP Guide for more information.

**extractProcess**

```cpp
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

void flush()

Ensure that accumulated data and control information are transmitted on an APPC mapped conversation.

Conditions

INVREQ, NOTALLOC

free

void free()

Return the APPC session to CICS so that it may be used by other tasks.

Conditions

INVREQ, NOTALLOC

get

virtual const IccBuf& get()

A synonym for receive. See "Polymorphic Behavior" on page 55 for information on polymorphism.

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

void issueConfirmation()

Sends positive response to a partner’s send request that specified the confirmation option.

Conditions
INVREQ, NOTALLOC, TERMERR, SIGNAL

issueError

void issueError()

Signals an error to the partner process.

Conditions
INVREQ, NOTALLOC, TERMERR, SIGNAL

issuePrepare

void issuePrepare()

This only applies to DTP over APPC links. It enables a syncpoint initiator to prepare a syncpoint slave for syncpointing by sending only the first flow (‘prepare to commit’) of the syncpoint exchange.

Conditions
INVREQ, NOTALLOC, TERMERR

issueSignal

void issueSignal()

Signals that a mode change is needed.

Conditions
INVREQ, NOTALLOC, TERMERR

PIPList

IccBuf& 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.

process

const IccBuf& process() const
IccSession

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

virtual void put(const IccBuf& data)

data
A reference to an IccBuf object that holds the data to be sent to the remote process.
A synonym for send. See "Polymorphic Behavior" on page 55 for information on polymorphism.

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)

void send(SendOpt option = normal)

option
An enumeration, defined in this class, that affects the behavior of the send method. The default is normal.
Sends data to the remote partner.

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.
**sendInvite (2)**

```c
void sendInvite(SendOpt option = normal)
```

**option**  
An enumeration, defined in this class, that affects the behavior of the `sendInvite` method. The default is normal.

**sendLast (1)**

```c
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)**

```c
void sendLast(SendOpt option = normal)
```

**option**  
An enumeration, defined in this class, that affects the behavior of the `sendLast` method. The default is normal.

**state**

```c
IccValue::CVDA state(StateOpt option = lastCommand)
```

**option**  
An enumeration, defined in this class, that indicates how to report the state of the conversation

Returns a CVDA, defined in `IccValue` structure, that indicates the current state of the APPC conversation. Possible values are:

- ALLOCATED
- CONFFREE
- CONFSEND

**Conditions**

INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

**state**

```c
IccValue::CVDA state(StateOpt option = lastCommand)
```

**option**  
An enumeration, defined in this class, that indicates how to report the state of the conversation

Returns a CVDA, defined in `IccValue` structure, that indicates the current state of the APPC conversation. Possible values are:

- ALLOCATED
- CONFFREE
- CONFSEND

**Conditions**

INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR
IccSession

FREE
PENDFREE
PENDRECEIVE
RECEIVE
ROLLBACK
SEND
SYNCFREE
SYNCRECEIVE
SYNCSEND
NOTAPPLIC

IccValue::NOTAPPLIC is returned if there is no APPC conversation state.

Conditions
INVREQ, NOTALLOC

stateText

const char* stateText(StateOpt option = lastCommand)

option
An enumeration, defined in this class, that indicates how to report the state of the conversation

Returns the symbolic name of the state that state method would return. For example, if state returns IccValue::ALLOCATED, stateText would return "ALLOCATED".

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
</tbody>
</table>
**Inherited protected methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

**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**

Used 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
IccSession
Chapter 47. IccStartRequestQ class

IccBase  
  IccResource  
  IccStartRequestQ

This is a singleton class that enables the application programmer to request an asynchronous start of another CICS transaction (see the start method on page 196).

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

cancel

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.

Cancels a previously issued start request that has not yet expired.

Conditions
ISCINVREQ, NOTAUTH, NOTFND, SYSIDERR

clearData

void clearData()

clearData clears the current data that is to be passed to the started transaction. 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**

```cpp
class IccStartRequestQ
{
public:
  const IccBuf& data() const
  {
    // Implementation
  }
}
```

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.

**instance**

```cpp
class IccStartRequestQ
{
public:
  static IccStartRequestQ* instance()
  {
    // Implementation
  }
}
```

Returns a pointer to the single IccStartRequestQ object. If the object does not exist it is created. See also **startRequestQ** method on page 105 of IccControl.

**queueName**

```cpp
class IccStartRequestQ
{
public:
  const char* queueName() const
  {
    // Implementation
  }
}
```

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**

```cpp
class IccStartRequestQ
{
public:
  void registerData(IccBuf* buffer)
  {
    // Implementation
  }
}
```

A pointer to the IccBuf object that holds data to be passed on a start request. 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.

**reset**

```cpp
class IccStartRequestQ
{
public:
  void reset()
  {
    // Implementation
  }
}
```

Clears any associations previously made by set... methods in this class.

**retrieveData**

```cpp
class IccStartRequestQ
{
public:
  void retrieveData(RetrieveOpt option = noWait)
  {
    // Implementation
  }
}
```

**option**

An enumeration, defined in this class, that indicates what happens if there is no start data available.

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.
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

const IccTermId& returnTermId() const

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.

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 194 for an alternative way to pass data to started transactions.

setQueueName

void setQueueName(const char* queueName)

queueName
An 8-character queue name.
Requests that this queue name be passed to the started transaction when the start method is called.

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)

void setReturnTermId(const char* termName)

termName
The 4-character name of the terminal that is involved in the session.
IccStartRequestQ

Requests that this return terminal ID be passed to the started transaction when the start method is called.

setReturnTransId (1)

void setReturnTransId(const IccTransId& transId)

transId
A reference to an IccTransId object.

setReturnTransId (2)

void setReturnTransId(const char* transName)

transName
The 4-character name of the return transaction.
Requests that this return transaction ID be passed to the started transaction when the start method is called.

setStartOpts

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.
Sets whether the started transaction is to have protection and whether it is to be checked.

start

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.

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.

Conditions
INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, SYSIDERR, TERMIDERR, TRANSIDERR, USERIDERR

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>isRouteOptionOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>routeOption</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
<tr>
<td>setRouteOption</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
IccStartRequestQ

Enumerations

RetrieveOpt
- noWait
- wait

ProtectOpt
- none
- protect

CheckOpt
- check
- noCheck
Chapter 48. IccSysId class

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)

IccSysId(const IccSysId& id)

id A reference to an IccSysId object.
The copy constructor.

Public methods

operator=(1)

IccSysId& operator=(const IccSysId& id)

id A reference to an existing IccSysId object.

operator=(2)

IccSysId& operator=(const char* name)

name
The 4-character name of the CICS system.
Sets the name of the CICS system held in the object.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 49. IccSystem class

IccBase
    IccResource
    IccSystem

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.

Header file: ICCSYSEH

Sample: ICC$SYS

IccSystem constructor (protected)

Constructor

IccSystem()

Public methods

applName

const char* applName()

Returns the 8-character name of the CICS region.

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)

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.
IccSystem

resName
The name of the resource that is to be the starting point for browsing the resources.
Signals the start of a browse through a set of CICS resources.

Conditions
END, FILENOTFOUND, ILLOGIC, NOTAUTH

dateFormat

const char* dateFormat()

Returns the default dateFormat for the CICS region.

Conditions
INVREQ

derBrowse

void endBrowse(ResourceType resource)

Signals the end of a browse through a set of CICS resources.

Conditions
END, FILENOTFOUND, ILLOGIC, NOTAUTH

freeStorage

void freeStorage(void* pStorage)

Releases the storage obtained by the IccSystem getStorage method.

Conditions
INVREQ

getFile (1)

IccFile* getFile(const IccFileId& id)

id A reference to an IccFileId object that identifies a CICS file.

ggetFile (2)

IccFile* getFile(const char* fileName)

fileName The name of a CICS file.
Returns a pointer to the IccFile object identified by the argument.

Conditions
END, FILENOTFOUND, ILLOGIC, NOTAUTH
getNextFile

IccFile* 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.

**Conditions**
END, FILENOTFOUND, ILLOGIC, NOTAUTH

getStorage

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.

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.

**Conditions**
LENGERR, NOSTG

instance

static IccSystem* instance()

Returns a pointer to the singleton IccSystem object. The object is created if it does not already exist.

operatingSystem

char operatingSystem()

Returns a 1-character value that identifies the operating system under which CICS is running:

A  AIX
N  Windows NT
P  OS/2
X  MVS/ESA

**Conditions**
NOTAUTH

operatingSystemLevel

unsigned short operatingSystemLevel()
IccSystem

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.

**Conditions**

NOTAUTH

release

```cpp
unsigned long release()
```

Returns the level of the CICS system as an integer set to 100 multiplied by the version number plus 10 multiplied by the release level. For example, CICS Transaction Server for z/OS [Version 1] Release 3 would return 130.

**Conditions**

NOTAUTH

releaseText

```cpp
const char* 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".

**Conditions**

NOTAUTH

sysId

```cpp
IccSysId& sysId()
```

Returns a reference to the `IccSysId` object that identifies this CICS system.

**Conditions**

INVREQ

workArea

```cpp
const IccBuf& workArea()
```

Returns a reference to the `IccBuf` object that holds the work area for the CICS system.

**Conditions**

INVREQ

## Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>Method</td>
<td>Class</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

**Inherited protected methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

**Enumerations**

**ResourceType**

- autoInstallModel
- connection
- dataQueue
- exitProgram
- externalDataSet
- file
- journal
- modename
- partner
- profile
- program
- requestld
- systemDumpCode
- tempStore
- terminal
- transactionDumpCode
- transaction
- transactionClass
IccSystem
Chapter 50. IccTask class

IccBase
  IccResource
  IccTask

IccTask is a singleton class used to invoke task related CICS services.

Header file: ICTXSKEH

Sample: ICC$TSK

IccTask Constructor (protected)

Constructor

IccTask()

Public methods

<table>
<thead>
<tr>
<th>The opt parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many methods have the same parameter, opt, which is described under the</td>
</tr>
<tr>
<td>abendCode method in &quot;abendCode&quot; on page 67</td>
</tr>
</tbody>
</table>

abend

```
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.

Requests CICS to abend this task.

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()
```
**IccTask**

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**

```cpp
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.

Requests that this task be delayed for an interval of time, or until a specific time.

**Conditions**
EXPIRED, INVREQ

**dump**

```cpp
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.

Requests CICS to take a dump for this task. (See also setDumpOpts.) Returns the character identifier of the dump.

**Conditions**
INVREQ, IOERR, NOSPACE, NOSTG, NOTOPEN, OPENERR, SUPPRESSED

**enterTrace**

```cpp
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.

Writes a user trace entry in the CICS trace table.

Conditions
   INVREQ, LENGERR

facilityType

FacilityType 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.

Conditions
   INVREQ

freeStorage

void freeStorage(void* pStorage)

Releases the storage obtained by the IccTask getStorage method.

Conditions
   INVREQ

getStorage

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.

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 203 in IccSystem class.

Conditions
   LENGERR, NOSTG
IccTask

instance

    static IccTask* instance();

    Returns a pointer to the singleton IccTask object. The object is created if it does not already exist.

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

Icc::Bool 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 158 in IccProgram).

    Conditions
    INVREQ

isResourceSecurityOn

Icc::Bool isResourceSecurityOn()

    Returns a boolean, defined in Icc structure, that indicates whether this task is subject to resource security checking.

    Conditions
    INVREQ

isRestarted

Icc::Bool isRestarted()

    Returns a boolean, defined in Icc structure, that indicates whether this task has been automatically restarted by CICS.

    Conditions
    INVREQ

isStartDataAvailable

Icc::Bool 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.
number

unsigned long number() const

Returns the number of this task, unique within the CICS system.

principalSysId

IccSysId& principalSysId(Icc::GetOpt opt = Icc::object)

Returns a reference to an IccSysId object that identifies the principal system identifier for this task.

priority

unsigned short priority(Icc::GetOpt opt = Icc::object)

Returns the priority for this task.

rollBackUOW

void rollBackUOW()

Roll back (backout) the resource updates associated with the current UOW within this task.

setDumpOpts

void setDumpOpts(unsigned long opts = dDefault)

 opts
 An integer, made by adding or logically ORing values from the DumpOpts enumeration, defined in this class.
 Set the dump options for this task. This method affects the behavior of the dump method defined in this class.

setPriority

void setPriority(unsigned short pri)

 pri
 The new priority.
 Changes the dispatch priority of this task.
IccTask

**Conditions**

INVREQ

**setWaitText**

```cpp
void setWaitText(const char* name)
```

*name*

The 8-character string label that indicates why this task is waiting. 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.

**startType**

```cpp
StartType startType()
```

Returns an enumeration, defined in this class, that indicates how this task was started.

**Conditions**

INVREQ

**suspend**

```cpp
void suspend()
```

Suspends this task, allowing other tasks to be dispatched.

**transId**

```cpp
const IccTransId& transId()
```

Returns the `IccTransId` object representing the transaction name of this CICS task.

**triggerDataQueueId**

```cpp
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**

```cpp
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.

Returns the ID of the user associated with this task.
Conditions
INVREQ

**waitExternal**

```c
void waitExternal (long** ECBList,
               unsigned long numEvents,
               WaitPurgeability opt = purgeable,
               WaitPostType type = MVSPost)
```

*ECBList*
A pointer to a list 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.

Waits for events that post ECBs - Event Control Blocks. 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.

See *waitExternal* in the *CICS Application Programming Reference* for more information about ECBs.

Conditions
INVREQ

**waitOnAlarm**

```c
void waitOnAlarm(const IccAlarmRequestId& id)
```

*id*  A reference to the *IccAlarmRequestId* object that identifies a particular alarm request.

Suspended the task until the alarm goes off (expires). See also *setAlarm* on page 93 in *IccClock*.

Conditions
INVREQ

**workArea**

```c
IccBuf& workArea()
```

Returns a reference to the *IccBuf* object that holds the work area for this task.

Conditions
INVREQ
Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

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 desired combination. For example IccTask::dProgram + IccTask::dDCT + IccTask::dSIT.
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 backend 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.
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.
IccTask

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.
Chapter 51. IccTempStore class

IccBase
IccResource
IccTempStore

IccTempStore objects are used to manage the temporary storage of data. (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

The opt parameter

Many methods have the same parameter, opt, which is described under the abendCode method inabendCode on page 67.

clear

virtual void clear()

A synonym for empty. See Polymorphic Behavior on page 55 for information on polymorphism.

empty
IccTempStore

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

virtual const IccBuf& get()

A synonym for readNextItem. See "Polymorphic Behavior" on page 55 for information on polymorphism.

numberOfItems

unsigned short numberOfItems() const

Returns the number of items in temporary storage. This is only valid after a successful writeItem call.

put

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.

A synonym for writeItem. See "Polymorphic Behavior" on page 55 for information on polymorphism.

readItem

const IccBuf& readItem(unsigned short itemNum)

itemNum
Specifies the item number of the logical record to be retrieved from the queue. Reads the specified item from the temporary storage queue and returns a reference to the IccBuf object that contains the information.

Conditions
INVREQ, IOERR, ISCINVREQ, ITEMERR, LENGERR, NOTAUTH, QIDERR, SYSIDERR

readNextItem

const IccBuf& readNextItem()

Reads the next item from a temporary storage queue and returns a reference to the IccBuf object that contains the information.

Conditions
INVREQ, IOERR, ISCINVREQ, ITEMERR, LENGERR, NOTAUTH, QIDERR, SYSIDERR
rewriteltem

```c
void rewriteltem (unsigned short itemNum,
                 const IccBuf& item,
                 NoSpaceOpt opt = suspend)
```

The parameters are:

- `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.

This method updates the specified item in the temporary storage queue.

**Conditions**

INVREQ, IOERR, ISCINVREQ, ITEMERR, LENGERR, NOSPACE, NOTAUTH, QIDERR, SYSIDERR

**writeltem (1)**

```c
unsigned short writeltem (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.

**writeltem (2)**

```c
unsigned short writeltem (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.

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).

**Conditions**

INVREQ, IOERR, ISCINVREQ, ITEMERR, LENGERR, NOSPACE, NOTAUTH, QIDERR, SYSIDERR
### Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>isRouteOptionOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>routeOption</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
<tr>
<td>setRouteOption</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

### Enumerations

#### Location

- **auxStorage**
  Temporary store data is to reside in auxiliary storage (disk).

- **memory**
  Temporary store data is to reside in memory.

#### NoSpaceOpt

What action to take if a shortage of space in the queue prevents the record being added immediately.

- **suspend**
  Suspends the application program.

- **returnCondition**
  Do not suspend the application program, but raise the NOSPACE condition instead.
Chapter 52. IccTempStoreId class

IccBase
   IccResourceId
      IccTempStoreId

IccTempStoreId class is used to identify a temporary storage name in the CICS system. This is an entry in the TST (temporary storage table).

Header file: ICCRIDEH

IccTempStoreId constructors

Constructor (1)

IccTempStoreId(const char* name)

name
   The 8-character name of the temporary storage entry.

Constructor (2)

IccTempStoreId(const IccTempStoreId& id)

id A reference to an IccTempStoreId object.
The copy constructor.

Public methods

operator= (1)

IccTempStoreId& operator=(const char* name)

name
   The 8-character name of the temporary storage entry.

operator= (2)

IccTempStoreId& operator=(const IccTempStoreId& id)

id A reference to an IccTempStoreId object.
Assigns a new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
### IccTempStoreId

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 53. IccTermId class

IccBase
   IccResourceId
     IccTermId

IccTermId class is used to identify a terminal name in the CICS system. This is an entry in the TCT (terminal control table).

Header file: ICCRIDEH

IccTermId constructors

Constructor (1)

IccTermId(const char* name)

name
   The 4-character name of the terminal

Constructor (2)

IccTermId(const IccTermId& id)

id
   A reference to an IccTermId object.
   The copy constructor.

Public methods

operator= (1)

IccTermId& operator=(const char* name)

name
   The 4-character name of the terminal

operator= (2)

IccTermId& operator=(const IccTermId& id)

id
   A reference to an IccTermId object.
   Assigns a new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1989, 2001
### IccTermId

<table>
<thead>
<tr>
<th><strong>Method</strong></th>
<th><strong>Class</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

#### Inherited protected methods

<table>
<thead>
<tr>
<th><strong>Method</strong></th>
<th><strong>Class</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

224  CICS Transaction Server: C++ OO Class Libraries
Chapter 54. IccTerminal class

IccBase
  IccResource
   IccTerminal

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.

Header file: ICCTRMEH

Sample: ICC$TRM

IccTerminal constructor (protected)

Constructor

IccTerminal()

Public methods

| The opt parameter | Many methods have the same parameter, opt, which is described under the abendCode method in “abendCode” on page 67 |

AID

AIDVal AID()

Returns an enumeration, defined in this class, that indicates which AID (action identifier) key was last pressed at this terminal.

clear

virtual void clear()

A synonym for erase. See “Polymorphic Behavior” on page 55 for information on polymorphism.

cursor

unsigned short cursor()

Returns the current cursor position as an offset from the top left corner of the screen.

data

IccTerminalData* data()
IccTerminal

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

dfreeKeyboard

void freeKeyboard()

Frees the keyboard so that the terminal can accept input.

Conditions
INVREQ, INVPARTN

get

virtual const IccBuf& get()

A synonym for receive. See "Polymorphic Behavior" on page 55 for information on polymorphism.

height

unsigned short height(Icc::getopt opt = Icc::object)

Returns how many lines the screen holds.

Conditions
INVREQ

inputCursor

unsigned short inputCursor()

Returns the position of the cursor on the screen.

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 top 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)
IccTerminal

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)


receive

const IccBuf& receive(Case caseOpt = upper)

caseOpt
  An enumeration, defined in this class, that indicates whether text is to be converted to upper case or left as it is.
  Receives data from the terminal

Conditions
  EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR
receive3270Data

const IccBuf& receive3270Data(Case caseOpt = upper)

caseOpt
   An enumeration, defined in this class, that indicates whether text is to be
   converted to upper case or left as it is.
Receives the 3270 data buffer from the terminal

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)

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.
**IccTerminal**

Writes the specified data to either the current cursor position or to the cursor position specified by the arguments.

**Conditions**

INVREQ, LENGERR, TERMERR

**send3270 (1)**

```cpp
void send3270(const IccBuf & buffer)
```

- **buffer**
  A reference to an **IccBuf** object that holds the data that is to be sent.

**send3270 (2)**

```cpp
void send3270 (const char* format, ...
```

- **format**
  A format string, as in the **printf** standard library function
  - The optional arguments that accompany **format**.

**send3270 (3)**

```cpp
void send3270 (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.

**send3270 (4)**

```cpp
void send3270 (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**.
Writes the specified data to either the next line of the terminal or to the specified column of the current line.

**Conditions**

INVREQ, LENGERR, TERMERR

**sendLine (1)**

```cpp
void sendLine(const IccBuf & buffer)
```

- **buffer**
  A reference to an **IccBuf** object that holds the data that is to be sent.
sendLine (2)

```c
void sendLine (const char* format, ...
```

*format*  
A format string, as in the `printf` standard library function

...  
The optional arguments that accompany `format`.

sendLine (3)

```c
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)

```c
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`.

Writes the specified data to either the next line of the terminal or to the specified column of the current line.

**Conditions**

INVREQ, LENGERR, TERMERR

setColor

```c
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.

Changes the color of the text subsequently sent to the terminal.

setCursor (1)

```c
void setCursor(unsigned short offset)
```

*offset*  
The position of the cursor where the top left corner is 0.
IccTerminal

setCursor (2)

```cpp
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

Two different ways of setting the position of the cursor on the screen.

**Conditions**
INVREQ, INVPARTN

setHighlight

```cpp
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.

Changes the highlighting of the data subsequently sent to the terminal.

setLine

```cpp
void setLine(unsigned short lineNum = 1)
```

**lineNum**
The line number, counting from the top.

Moves the cursor to the start of line *lineNum*, where 1 is the top line of the terminal.
The default is to move the cursor to the start of line 1.

**Conditions**
INVREQ, INVPARTN

setNewLine

```cpp
void setNewLine(unsigned short numLines = 1)
```

**numLines**
The number of blank lines.

Requests that *numLines* blank lines be sent to the terminal.

**Conditions**
INVREQ, INVPARTN

setNextCommArea

```cpp
void setNextCommArea(const IccBuf& commArea)
```

**commArea**
A reference to the buffer that is to be used as a COMMAREA.
Specifies the COMMAREA that is to be passed to the next transaction started on this terminal.
**IccTerminal**

**setNextInputMessage**

```cpp
void setNextInputMessage(const IccBuf& message)
```

*message*

A reference to the buffer that holds the input message.

Specifies data that is to be made available, by the `receive` method, to the next transaction started at this terminal.

**setNextTransId**

```cpp
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.

Specifies the next transaction that is to be started on this terminal.

**signoff**

```cpp
void signoff()
```

Signs off the user who is currently signed on. Authority reverts to the default user.

**Conditions**

INVREQ

**signon (1)**

```cpp
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)**

```cpp
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.

Signs the user on to the terminal.

Conditions
 INVREQ, NOTAUTH, USERIDERR

waitForAID (1)

AIDVal waitForAID()

Waits for any input and returns an enumeration, defined in this class, that indicates which AID key is expected.

waitForAID (2)

void waitForAID(AIDVal aid)

aid

An enumeration, defined in this class, that indicates which AID key was last pressed.

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.

Conditions
 EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

width

unsigned short width(Icc::getopt opt = Icc::object)

Returns the width of the screen in characters.

Conditions
 INVREQ

workArea

IccBuf& workArea()

Returns a reference to the IccBuf object that holds the terminal work area.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
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

Method Class
setClassName IccBase
setCustomClassNum IccBase

Enumerations

AIDVal
ENETER  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.

IccTerminal
**immediate**

Start the transaction immediately, that is, before any other outstanding starts queued on the terminal.
Chapter 55. IccTerminalData class

IccBase
   IccResource
   IccTerminalData

IccTerminalData is a singleton class owned by IccTerminal (see data on page 225 in IccTerminal class). IccTerminalData contains information about the terminal characteristics.

Header file: ICCTMDEH

Sample: ICC$TRM

IccTerminalData constructor (protected)

Constructor

IccTerminalData()

Public methods

The opt parameter

Many methods have the same parameter, opt, which is described under the abendCode method in "abendCode" on page 67.

alternateHeight

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. Returns the alternate height of the screen, in lines.

Conditions

INVREQ

alternateWidth

unsigned short alternateWidth(Icc::GetOpt opt = Icc::object)

Returns the alternate width of the screen, in characters.

Conditions

INVREQ

defaultHeight

unsigned short defaultHeight(Icc::GetOpt opt = Icc::object)

Returns the default height of the screen, in lines.
IccTerminalData

**Conditions**

INVREQ

defaultWidth

```cpp
unsigned short defaultWidth(Icc::GetOpt opt = Icc::object)
```

Returns the default width of the screen, in characters.

**Conditions**

INVREQ

graphicCharCodeSet

```cpp
unsigned short graphicCharCodeSet(Icc::GetOpt opt = Icc::object)
```

Returns the binary code page global identifier as a value in the range 1 to 65534, or 0 for a non-graphics terminal.

**Conditions**

INVREQ

graphicCharSetId

```cpp
unsigned short graphicCharSetId(Icc::GetOpt opt = Icc::object)
```

Returns the graphic character set global identifier as a number in the range 1 to 65534, or 0 for a non-graphics terminal.

**Conditions**

INVREQ

isAPLKeyboard

```cpp
Icc::Bool isAPLKeyboard(Icc::GetOpt opt = Icc::object)
```

Returns a boolean that indicates whether the terminal has the APL keyboard feature.

**Conditions**

INVREQ

isAPLText

```cpp
Icc::Bool isAPLText(Icc::GetOpt opt = Icc::object)
```

Returns a boolean that indicates whether the terminal has the APL text feature.

**Conditions**

INVREQ

isBTrans

```cpp
Icc::Bool isBTrans(Icc::GetOpt opt = Icc::object)
```
Returns a boolean that indicates whether the terminal has the background transparency capability.

**Conditions**
INVREQ

### isColor

```cpp
Icc::Bool isColor(Icc::GetOpt opt = Icc::object)
```

Returns a boolean that indicates whether the terminal has the extended color capability.

**Conditions**
INVREQ

### isEWA

```cpp
Icc::Bool isEWA(Icc::GetOpt opt = Icc::object)
```

Returns a Boolean that indicates whether the terminal supports Erase Write Alternative.

**Conditions**
INVREQ

### isExtended3270

```cpp
Icc::Bool isExtended3270(Icc::GetOpt opt = Icc::object)
```

Returns a Boolean that indicates whether the terminal supports the 3270 extended data stream.

**Conditions**
INVREQ

### isFieldOutline

```cpp
Icc::Bool isFieldOutline(Icc::GetOpt opt = Icc::object)
```

Returns a boolean that indicates whether the terminal supports field outlining.

**Conditions**
INVREQ

### isGoodMorning

```cpp
Icc::Bool isGoodMorning(Icc::GetOpt opt = Icc::object)
```

Returns a boolean that indicates whether the terminal has a ‘good morning’ message.

**Conditions**
INVREQ
IccTerminalData

isHighlight

Icc::Bool isHighlight(Icc::GetOpt opt = Icc::object)

Returns a boolean that indicates whether the terminal has extended highlight capability.

Conditions
INVREQ

isKatakana

Icc::Bool isKatakana(Icc::GetOpt opt = Icc::object)

Returns a boolean that indicates whether the terminal supports Katakana.

Conditions
INVREQ

isMSRControl

Icc::Bool isMSRControl(Icc::GetOpt opt = Icc::object)

Returns a boolean that indicates whether the terminal supports magnetic slot reader control.

Conditions
INVREQ

isPS

Icc::Bool isPS(Icc::GetOpt opt = Icc::object)

Returns a boolean that indicates whether the terminal supports programmed symbols.

Conditions
INVREQ

isSOSI

Icc::Bool isSOSI(Icc::GetOpt opt = Icc::object)

Returns a boolean that indicates whether the terminal supports mixed EBCDIC/DBCS fields.

Conditions
INVREQ

isTextKeyboard

Icc::Bool isTextKeyboard(Icc::GetOpt opt = Icc::object)

Returns a boolean that indicates whether the terminal supports TEXTKYBD.
Conditions
INVREQ

isTextPrint

Icc::Bool isTextPrint(Icc::GetOpt opt = Icc::object)

Returns a boolean that indicates whether the terminal supports TEXTPRINT.

Conditions
INVREQ

isValidation

Icc::Bool isValidation(Icc::GetOpt opt = Icc::object)

Returns a boolean that indicates whether the terminal supports validation.

Conditions
INVREQ

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
IccTerminalData
Chapter 56. IccTime class

IccBase
   IccResource
      IccTime

IccTime is used to contain time information and is the base class from which IccAbsTime, IccTimeInterval, and IccTimeOfDay classes are derived.

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

hours

virtual unsigned long hours() const

Returns the hours component of time—the value specified in the constructor.

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.
IccTime

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

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.
IccTime
Chapter 57. IccTimeInterval class

IccBase
   IccResource
      IccTime
         IccTimeInterval

This class holds information about a time interval.

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)

IccTimeInterval(const IccTimeInterval& time)

The copy constructor.

Public methods

operator=

IccTimeInterval& operator=(const IccTimeInterval& timeInterval)

Assigns one IccTimeInterval object to another.

set

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
IccTimeInterval

Changes the time held in the IccTimeInterval object.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccResource</td>
</tr>
<tr>
<td>className</td>
<td>IccResource</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>hours</td>
<td>IccTime</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>minutes</td>
<td>IccTime</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
<tr>
<td>timeInHours</td>
<td>IccTime</td>
</tr>
<tr>
<td>timeInMinutes</td>
<td>IccTime</td>
</tr>
<tr>
<td>timeInSeconds</td>
<td>IccTime</td>
</tr>
<tr>
<td>type</td>
<td>IccTime</td>
</tr>
</tbody>
</table>

Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 58. IccTimeOfDay class

```
IccBase
   IccResource
      IccTime
         IccTimeOfDay
```

This class holds information about the time of day.

**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)**

```
IccTimeOfDay(const IccTimeOfDay& time)
```

The copy constructor

### Public methods

**operator=**

```
IccTimeOfDay& operator=(const IccTimeOfDay& timeOfDay)
```

Assigns one `IccTimeOfDay` object to another.

**set**

```
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
IccTimeOfDay

Changes the time held in the **IccTimeOfDay** object.

### Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>hours</td>
<td>IccTime</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>minutes</td>
<td>IccTime</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
<tr>
<td>timeInHours</td>
<td>IccTime</td>
</tr>
<tr>
<td>timeInMinutes</td>
<td>IccTime</td>
</tr>
<tr>
<td>timeInSeconds</td>
<td>IccTime</td>
</tr>
<tr>
<td>type</td>
<td>IccTime</td>
</tr>
</tbody>
</table>

### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 59. IccTPNameId class

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)

IccTPNameId(const IccTPNameId& id)

id A reference to an IccTPNameId object.
The copy constructor.

Public methods

operator= (1)

IccTPNameId& operator=(const char* name)

name
   The 1- to 64-character TP name.

operator= (2)

IccTPNameId& operator=(const IccTPNameId& id)

id A reference to an IccTPNameId object.
Assigns a new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 60. IccTransId class

IccBase
 IccResourceld
 IccTransId

IccTransId class identifies a transaction name in the CICS system. This is an entry in the PCT (Program Control Table).

**Header file**: ICCRIDEH

IccTransId constructors

**Constructor (1)**

IccTransId(const char* name)

*name*  
The 4-character transaction name.

**Constructor (2)**

IccTransId(const IccTransId& id)

*id*  
A reference to an IccTransId object.  
The copy constructor.

Public methods

**operator= (1)**

IccTransId& operator=(const char* name)

*name*  
The 4-character transaction name.

**operator= (2)**

IccTransId& operator=(const IccTransId& id)

*id*  
A reference to an IccTransId object.  
Assigns a new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceld</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceld</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1989, 2001
**IccTransId**

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

**Inherited protected methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 61. IccUser class

IccBase
   IccResource
      IccUser

This class represents a CICS user.

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

groupName  The optional 8-character group ID.

Public methods

changePassword

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.
Attempts to change the user's password.

Conditions
INVREQ, NOTAUTH, USERIDERR
IccUser

daysUntilPasswordExpires

    unsigned short daysUntilPasswordExpires() const

Returns the number of days before the password expires. This method is valid after
a successful verifyPassword method call in this class.

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
IccUser

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**

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionOnConditionAsChar</td>
<td>IccResource</td>
</tr>
<tr>
<td>actionsOnConditionsText</td>
<td>IccResource</td>
</tr>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>condition</td>
<td>IccResource</td>
</tr>
<tr>
<td>conditionText</td>
<td>IccResource</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>handleEvent</td>
<td>IccResource</td>
</tr>
<tr>
<td>id</td>
<td>IccResource</td>
</tr>
<tr>
<td>isEDFOn</td>
<td>IccResource</td>
</tr>
<tr>
<td>name</td>
<td>IccResource</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
<tr>
<td>setActionOnAnyCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionOnCondition</td>
<td>IccResource</td>
</tr>
<tr>
<td>setActionsOnConditions</td>
<td>IccResource</td>
</tr>
<tr>
<td>setEDF</td>
<td>IccResource</td>
</tr>
</tbody>
</table>
Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
Chapter 62. IccUserId class

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)

IccUserId(const IccUserId& id)

id  A reference to an IccUserId object.
The copy constructor.

Public methods

operator= (1)

IccUserId& operator=(const char* name)

name
The 8-character name of the user ID.

operator= (2)

IccUserId& operator=(const IccUserId& id)

id  A reference to an IccUserId object.
    Assigns a new value.

Inherited public methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>classType</td>
<td>IccBase</td>
</tr>
<tr>
<td>className</td>
<td>IccBase</td>
</tr>
<tr>
<td>customClassNum</td>
<td>IccBase</td>
</tr>
<tr>
<td>name</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>nameLength</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>operator delete</td>
<td>IccBase</td>
</tr>
<tr>
<td>operator new</td>
<td>IccBase</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1989, 2001
### Inherited protected methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>IccResourceId</td>
</tr>
<tr>
<td>setClassName</td>
<td>IccBase</td>
</tr>
<tr>
<td>setCustomClassNum</td>
<td>IccBase</td>
</tr>
</tbody>
</table>
**Chapter 63. IccValue structure**

This structure contains CICS-value data areas (CVDAs) as an enumeration.

**Header file:** ICCVALEH

### Enumeration

**CVDA**

Valid CVDAs are:

| CVDA     | ACQFAIL  | ADD      | ALARM    | ALTERABLE | APLKBTD  | APPCSINGLE | ASCI8     | AUDALARM  | AUTOINACTIVE | AUXPAUSE  | BACKOUT   | BASESPACE  | BELOW     | BLK       | BTAM      | CANCEL    | CEDF      | CICSSECURITY | CLOSEFAILED | CMDPROT   | COBOL     | COLDSTART  | CONFFREE  | CONNECTION | CONTROLSHUT | COPY      | CTGMODIFY | DAE       | DATASTREAM | DEFAULT   | DEFRESPP3 | DELEXITERROR | DISABLED   | DISK1     | DPLSUBSET | DYNAMIC   | EMPTYREQ | EVENT     | EXCI      | EXITTRACE | FAILED   | FILE      |
|----------|----------|----------|----------|-----------|----------|------------|-----------|-----------|--------------|-----------|-----------|------------|-----------|-----------|-----------|-----------|----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
|          | ACQUIRED | ADDABLE  | ALLOCATED| ALTERNATE | APTEXT   | APPLICATION | ASSEMBLER | AUTOACTIVE | AUTOPAGEABLE | AUXSTART | BACKTRANS | BATCHLU    | BGAM      | BLOCKED  | BUSY      | CANCEL    | CICS      | CICSTABLE  | CLOSELEAVE | CMDSEC   | COBOLI1   | COLOR     | CONFEREIVE| CONNECTED | CONVERSE  | COPY      | CREATE    | CTIGNONE  | DATE      | DATA      | DB2®       | DEFRESP1  | DEFRESP1OR2| DEFRESP2  | DELETABLE  | DEREGEROR | DISABLING | DISK2     | DS3270    | DUALCASE  | DUMMY     | EMERGENCY | ENDAFFINITY | EXCEPT    | EXECNQ     | EXITED    | EXTRANDEDS| FAILEDBKOUT| FAILEDBKOUT| FAILINGBKOUT| FINPUT    | FIRSTINIT |
|          | ACQUIRING| ADDFAIL  | ALLQUERY | ALTPRTCOPY| APPC     | ASACTL     | ASSEMBLER | AUTOARCH   | AUTOSTART   | AUXSTOP  | BASEUPNONBWO | BDAM      | BIPROG    | BROWSABLE | C         | CD        | CICSDATAKEY| CLEAR      | CMDSCN    | COLAQ     | COMMIT    | CONFSEND   | CONSOLE   | CONNECT    | CONVERGE  | CRITICAL  | CTRLABLE  | DATASET   | DEADLOCK  | DEFRESP2  | DELETAEL  | DEREGEROR | DISABLING | DISK2PAUSE| DUALCASE  | DUMMY     | EMERGENCY | ENDAFFINITY | EXCEPT    | EXECNQ     | EXITED    | EXTENDEDDEDS| FAILEDBKOUT| FAILEDBKOUT| FACINGBKOUT| FINPUT    | FIRSTINIT |
|          | ACTIVE   | ADVANCE  | ALLQUERY | ANY       | APPCPARALLEL | ASCII7     | ATI       | ATTENTION  | AUXILIARY   | AVAILABLE | BASE       | BEGINSESSION | BSYNCH    | BSAM      | CACHE     | CDRDLPRT  | CICSEXECKEY | CLOSED     | CMDSECYES | COLOQUERY | COCOMIT    | CONTNLU    | CORETRUN   | CURRENT   | CTLGALL    | CURRENT   | CATALOG   | CURRENT   | DATASETFULL| DEC       | DEFRESP2  | DELETAEL  | DEREGEROR | DISABLING | DISK2PAUSE| DUALCASE  | DUMMY     | EMERGENCY | ENDAFFINITY | EXCEPT    | EXECNQ     | EXITED    | EXTENDEDDEDS| FAILEDBKOUT| FAILEDBKOUT| FACINGBKOUT| FINPUT    | FIRSTINIT |

© Copyright IBM Corp. 1989, 2001

261
Chapter 63. IccValue structure 263
<table>
<thead>
<tr>
<th>IccValue</th>
<th>TRANIDONLY</th>
<th>TSQ</th>
<th>TTCAM</th>
<th>TTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWX3335</td>
<td>UNCOMMITTED</td>
<td>UNAVAILABLE</td>
<td>UNBLOCKED</td>
<td></td>
</tr>
<tr>
<td>UNENABLING</td>
<td>UNCONNECTED</td>
<td>UNDEFINED</td>
<td>UNDETERMINED</td>
<td></td>
</tr>
<tr>
<td>UNQUIESCED</td>
<td>UNENABLING</td>
<td>UNKNOWN</td>
<td>UNPROTECTED</td>
<td></td>
</tr>
<tr>
<td>UPDATABLE</td>
<td>UNREGISTERED</td>
<td>UNSOLDATA</td>
<td>UOW</td>
<td></td>
</tr>
<tr>
<td>USEROFF</td>
<td>USER</td>
<td>USERDATAKEY</td>
<td>USEREXECKEY</td>
<td></td>
</tr>
<tr>
<td>VALIDATION</td>
<td>VARIABLE</td>
<td>VFORM</td>
<td>VIDEOTERM</td>
<td></td>
</tr>
<tr>
<td>VRRDS</td>
<td>VSAM</td>
<td>VTAM®</td>
<td>WAIT</td>
<td></td>
</tr>
<tr>
<td>WAITCOMMIT</td>
<td>WAITER</td>
<td>WAITFORGET</td>
<td>WAITING</td>
<td>XCF</td>
</tr>
<tr>
<td>WAITRMI</td>
<td>WARMSTART</td>
<td>WIN</td>
<td>XCPTRACE</td>
<td></td>
</tr>
<tr>
<td>XM</td>
<td>XNOTDONE</td>
<td>XOK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 64. 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, provided you know what you are doing, 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 page 57)
   • Family Subset enforcement (see page 66)
   • EDF enablement (see page 46)
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 functional part of the main code is shown below.

```cpp
void main(void)
{
    Icc::initializeEnvironment(ICC_CLASS_MEMORY_MGMT,
                               ICC_FAMILY_SUBSET,
                               ICC_EDF_BOOL);

    try
    {
        ICC_USER_CONTROL control;
        control.run();
    }
    catch(IccException& exc)
    {
        Icc::catchException(exc);
    }
    catch(...)
    {
        Icc::unknownException();
    }
    Icc::returnToCICS();
}
```

This is the main C++ entry point.
This call initializes the environment and is essential. The three parameters have previously been defined to the defaults for the platform.
main function

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.
Part 4. Appendixes
## Appendix A. 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.

<table>
<thead>
<tr>
<th>EXEC CICS</th>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABEND</td>
<td>IccTask</td>
<td>abend</td>
</tr>
<tr>
<td>ADDRESS COMMMAREA</td>
<td>IccControl</td>
<td>commArea</td>
</tr>
<tr>
<td>ADDRESS CWA</td>
<td>IccSystem</td>
<td>workArea</td>
</tr>
<tr>
<td>ADDRESS EIB</td>
<td>No direct access to EIB: please use appropriate method on appropriate class.</td>
<td></td>
</tr>
<tr>
<td>ADDRESS TCTUA</td>
<td>IccTerminal</td>
<td>workArea</td>
</tr>
<tr>
<td>ADDRESS TWA</td>
<td>IccTask</td>
<td>workArea</td>
</tr>
<tr>
<td>ALLOCATE</td>
<td>IccSession</td>
<td>allocate</td>
</tr>
<tr>
<td>ASKTIME</td>
<td>IccClock</td>
<td>update</td>
</tr>
<tr>
<td>ASSIGN ABCODE</td>
<td>IccAbendData</td>
<td>abendCode</td>
</tr>
<tr>
<td>ASSIGN ABDUMP</td>
<td>IccAbendData</td>
<td>isDumpAvailable</td>
</tr>
<tr>
<td>ASSIGN ABPROGRAM</td>
<td>IccAbendData</td>
<td>programName</td>
</tr>
<tr>
<td>ASSIGN ALTSCRNH</td>
<td>IccTerminalData</td>
<td>alternateHeight</td>
</tr>
<tr>
<td>ASSIGN ALTSCRNW</td>
<td>IccTerminalData</td>
<td>alternateWidth</td>
</tr>
<tr>
<td>ASSIGN APLKYBD</td>
<td>IccTerminalData</td>
<td>isAPLKeyboard</td>
</tr>
<tr>
<td>ASSIGN APLTEXT</td>
<td>IccTerminalData</td>
<td>isAPLText</td>
</tr>
<tr>
<td>ASSIGN ASRAINTRPT</td>
<td>IccAbendData</td>
<td>ASRAInterrupt</td>
</tr>
<tr>
<td>ASSIGN ASRAKEY</td>
<td>IccAbendData</td>
<td>ASRAKeyType</td>
</tr>
<tr>
<td>ASSIGN ASRAPSW</td>
<td>IccAbendData</td>
<td>ASRAPSW</td>
</tr>
<tr>
<td>ASSIGN ASRAREGS</td>
<td>IccAbendData</td>
<td>ASRARegisters</td>
</tr>
<tr>
<td>ASSIGN ASRASPC</td>
<td>IccAbendData</td>
<td>ASRAStorageType</td>
</tr>
<tr>
<td>ASSIGN ASRASTG</td>
<td>IccAbendData</td>
<td>ASRARSpaceType</td>
</tr>
<tr>
<td>ASSIGN APPLID</td>
<td>IccSystem</td>
<td>appName</td>
</tr>
<tr>
<td>ASSIGN BTRANS</td>
<td>IccTerminalData</td>
<td>isBTrans</td>
</tr>
<tr>
<td>ASSIGN CMDSEC</td>
<td>IccTask</td>
<td>isCommandSecurityOn</td>
</tr>
<tr>
<td>ASSIGN COLOR</td>
<td>IccTerminalData</td>
<td>isColor</td>
</tr>
<tr>
<td>ASSIGN CWALENG</td>
<td>IccSystem</td>
<td>workArea</td>
</tr>
<tr>
<td>ASSIGN DEFSCRNHT</td>
<td>IccTerminalData</td>
<td>defaultHeight</td>
</tr>
<tr>
<td>ASSIGN DEFSCRNW</td>
<td>IccTerminalData</td>
<td>defaultWidth</td>
</tr>
<tr>
<td>ASSIGN EWASUPP</td>
<td>IccTerminalData</td>
<td>isEWA</td>
</tr>
<tr>
<td>ASSIGN EXTDS</td>
<td>IccTerminalData</td>
<td>isExtended3270</td>
</tr>
<tr>
<td>ASSIGN FACILITY</td>
<td>IccTerminal</td>
<td>name</td>
</tr>
<tr>
<td>ASSIGN FCI</td>
<td>IccTask</td>
<td>facilityType</td>
</tr>
<tr>
<td>ASSIGN GCHARS</td>
<td>IccTerminalData</td>
<td>graphicCharSetId</td>
</tr>
<tr>
<td>ASSIGN GCODES</td>
<td>IccTerminalData</td>
<td>graphicCharCodeSet</td>
</tr>
</tbody>
</table>
## EXEC CICS to Foundation Class methods

<table>
<thead>
<tr>
<th>EXEC CICS</th>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGN GMMI</td>
<td>IccTerminalData</td>
<td>isGoodMorning</td>
</tr>
<tr>
<td>ASSIGN HILIGHT</td>
<td>IccTerminalData</td>
<td>isHighlight</td>
</tr>
<tr>
<td>ASSIGN INITPARM</td>
<td>IccControl</td>
<td>initData</td>
</tr>
<tr>
<td>ASSIGN INITPARMLEN</td>
<td>IccControl</td>
<td>initData</td>
</tr>
<tr>
<td>ASSIGN INVOKINGPROG</td>
<td>IccControl</td>
<td>callingProgramId</td>
</tr>
<tr>
<td>ASSIGN KATAKANA</td>
<td>IccTerminalData</td>
<td>isKatakana</td>
</tr>
<tr>
<td>ASSIGN NETNAME</td>
<td>IccTerminal</td>
<td>netName</td>
</tr>
<tr>
<td>ASSIGN OUTLINE</td>
<td>IccTerminalData</td>
<td>isFieldOutline</td>
</tr>
<tr>
<td>ASSIGN ORGABCODE</td>
<td>IccAbendData</td>
<td>originalAbendCode</td>
</tr>
<tr>
<td>ASSIGN PRINSYSID</td>
<td>IccTask</td>
<td>principalSysId</td>
</tr>
<tr>
<td>ASSIGN PROGRAM</td>
<td>IccControl</td>
<td>programId</td>
</tr>
<tr>
<td>ASSIGN PS</td>
<td>IccTerminalData</td>
<td>isPS</td>
</tr>
<tr>
<td>ASSIGN QNAME</td>
<td>IccTask</td>
<td>triggerDataQueueId</td>
</tr>
<tr>
<td>ASSIGN RESSEC</td>
<td>IccTask</td>
<td>isResourceSecurityOn</td>
</tr>
<tr>
<td>ASSIGN RESTART</td>
<td>IccTask</td>
<td>isRestarted</td>
</tr>
<tr>
<td>ASSIGN SCRNHT</td>
<td>IccTerminal</td>
<td>height</td>
</tr>
<tr>
<td>ASSIGN SCRNWD</td>
<td>IccTerminal</td>
<td>width</td>
</tr>
<tr>
<td>ASSIGN SOSI</td>
<td>IccTerminalData</td>
<td>isSOSI</td>
</tr>
<tr>
<td>ASSIGN STARTCODE</td>
<td>IccTask</td>
<td>startType,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>isCommitSupported,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>isStartDataAvailable</td>
</tr>
<tr>
<td>ASSIGN SYSID</td>
<td>IccSystem</td>
<td>sysId</td>
</tr>
<tr>
<td>ASSIGN TASKPRIORITY</td>
<td>IccTask</td>
<td>priority</td>
</tr>
<tr>
<td>ASSIGN TCTUALENG</td>
<td>IccTerminal</td>
<td>workArea</td>
</tr>
<tr>
<td>ASSIGN TEXTKYBD</td>
<td>IccTerminalData</td>
<td>isTextKeyboard</td>
</tr>
<tr>
<td>ASSIGN TEXTPRINT</td>
<td>IccTerminalData</td>
<td>isTextPrint</td>
</tr>
<tr>
<td>ASSIGN TWALENG</td>
<td>IccTask</td>
<td>workArea</td>
</tr>
<tr>
<td>ASSIGN USERID</td>
<td>IccTask</td>
<td>userId</td>
</tr>
<tr>
<td>ASSIGN VALIDATION</td>
<td>IccTerminalData</td>
<td>isValidation</td>
</tr>
<tr>
<td>CANCEL</td>
<td>IccClock</td>
<td>cancelAlarm</td>
</tr>
<tr>
<td>CANCEL</td>
<td>IccStartRequestQ</td>
<td>cancel</td>
</tr>
<tr>
<td>CHANGE PASSWORD</td>
<td>IccUser</td>
<td>changePassword</td>
</tr>
<tr>
<td>CHANGE TASK</td>
<td>IccTask</td>
<td>setPriority</td>
</tr>
<tr>
<td>CONNECT PROCESS</td>
<td>IccSession</td>
<td>connectProcess</td>
</tr>
<tr>
<td>CONVERSE</td>
<td>IccSession</td>
<td>converse</td>
</tr>
<tr>
<td>DELAY</td>
<td>IccTask</td>
<td>delay</td>
</tr>
<tr>
<td>DELETE</td>
<td>IccFile</td>
<td>deleteRecord</td>
</tr>
<tr>
<td>DELETE</td>
<td>IccFile</td>
<td>deleteLockedRecord</td>
</tr>
<tr>
<td>DELETEQ TD</td>
<td>IccDataQueue</td>
<td>empty</td>
</tr>
<tr>
<td>DELETEQ TS</td>
<td>IccTempStore</td>
<td>empty</td>
</tr>
<tr>
<td>DEQ</td>
<td>IccSemaphore</td>
<td>unlock</td>
</tr>
</tbody>
</table>
## EXEC CICS to Foundation Class methods

<table>
<thead>
<tr>
<th>EXEC CICS</th>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUMP TRANSACTION</td>
<td>IccTask</td>
<td>dump</td>
</tr>
<tr>
<td>DUMP TRANSACTION</td>
<td>IccTask</td>
<td>setDumpOpts</td>
</tr>
<tr>
<td>ENDBR</td>
<td>IccFileIterator</td>
<td>IccFileIterator (destructor)</td>
</tr>
<tr>
<td>ENQ</td>
<td>IccSemaphore</td>
<td>lock</td>
</tr>
<tr>
<td>ENQ</td>
<td>IccSemaphore</td>
<td>tryLock</td>
</tr>
<tr>
<td>ENTER TRACENUM</td>
<td>IccTask</td>
<td>enterTrace</td>
</tr>
<tr>
<td>EXTRACT ATTRIBUTES</td>
<td>IccSession</td>
<td>state, stateText</td>
</tr>
<tr>
<td>EXTRACT PROCESS</td>
<td>IccSession</td>
<td>extractProcess</td>
</tr>
<tr>
<td>FORMATTIME YYDDD, YYMMDD, etc</td>
<td>IccClock</td>
<td>date</td>
</tr>
<tr>
<td>FORMATTIME DATE</td>
<td>IccClock</td>
<td>date</td>
</tr>
<tr>
<td>FORMATTIME DATEFORM</td>
<td>IccSystem</td>
<td>dateFormat</td>
</tr>
<tr>
<td>FORMATTIME DAYCOUNT</td>
<td>IccClock</td>
<td>daysSince1900</td>
</tr>
<tr>
<td>FORMATTIME DAYOFWEEK</td>
<td>IccClock</td>
<td>dayOfWeek</td>
</tr>
<tr>
<td>FORMATTIME DAYOFMONTH</td>
<td>IccClock</td>
<td>dayOfMonth</td>
</tr>
<tr>
<td>FORMATTIME MONTHOFYEAR</td>
<td>IccClock</td>
<td>monthOfYear</td>
</tr>
<tr>
<td>FORMATTIME TIME</td>
<td>IccClock</td>
<td>time</td>
</tr>
<tr>
<td>FORMATTIME YEAR</td>
<td>IccClock</td>
<td>year</td>
</tr>
<tr>
<td>FREE</td>
<td>IccSession</td>
<td>free</td>
</tr>
<tr>
<td>FREEMAIN</td>
<td>IccTask</td>
<td>freeStorage</td>
</tr>
<tr>
<td>GETMAIN</td>
<td>IccTask</td>
<td>getStorage</td>
</tr>
<tr>
<td>HANDLE ABEND</td>
<td>IccControl</td>
<td>setAbendHandler, cancelAbendHandler, resetAbendHandler</td>
</tr>
<tr>
<td>INQUIRE FILE ACCESSMETHOD</td>
<td>IccFile</td>
<td>accessMethod</td>
</tr>
<tr>
<td>INQUIRE FILE ADD</td>
<td>IccFile</td>
<td>isAddable</td>
</tr>
<tr>
<td>INQUIRE FILE BROWSE</td>
<td>IccFile</td>
<td>isBrowseable</td>
</tr>
<tr>
<td>INQUIRE FILE DELETE</td>
<td>IccFileControl</td>
<td>isDeletable</td>
</tr>
<tr>
<td>INQUIRE FILE EMPTYSSTATUS</td>
<td>IccFile</td>
<td>isEmptyOn</td>
</tr>
<tr>
<td>INQUIRE FILE ENABLESTATUS</td>
<td>IccFile</td>
<td>enableStatus</td>
</tr>
<tr>
<td>INQUIRE FILE KEYPOSITION</td>
<td>IccFile</td>
<td>keyPosition</td>
</tr>
<tr>
<td>INQUIRE FILE OPENSTATUS</td>
<td>IccFile</td>
<td>openStatus</td>
</tr>
<tr>
<td>INQUIRE FILE READ</td>
<td>IccFile</td>
<td>isReadable</td>
</tr>
<tr>
<td>INQUIRE FILE RECORDFORMAT</td>
<td>IccFile</td>
<td>recordFormat</td>
</tr>
<tr>
<td>INQUIRE FILE RECORDSIZE</td>
<td>IccFile</td>
<td>recordLength</td>
</tr>
</tbody>
</table>
## EXEC CICS to Foundation Class methods

<table>
<thead>
<tr>
<th>EXEC CICS</th>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>INQUIRE FILE RECOVSTATUS</td>
<td>IccFile</td>
<td>isRecoverable</td>
</tr>
<tr>
<td>INQUIRE FILE TYPE</td>
<td>IccFile</td>
<td>type</td>
</tr>
<tr>
<td>INQUIRE FILE UPDATE</td>
<td>IccFile</td>
<td>isUpdatable</td>
</tr>
<tr>
<td>ISSUE ABEND</td>
<td>IccSession</td>
<td>issueAbend</td>
</tr>
<tr>
<td>ISSUE CONFIRMATION</td>
<td>IccSession</td>
<td>issueConfirmation</td>
</tr>
<tr>
<td>ISSUE ERROR</td>
<td>IccSession</td>
<td>issueError</td>
</tr>
<tr>
<td>ISSUE PREPARE</td>
<td>IccSession</td>
<td>issuePrepare</td>
</tr>
<tr>
<td>ISSUE SIGNAL</td>
<td>IccSession</td>
<td>issueSignal</td>
</tr>
<tr>
<td>LINK</td>
<td>IccProgram</td>
<td>link</td>
</tr>
<tr>
<td>LINK INPUTMSG INPUTMSGLEN</td>
<td>IccProgram</td>
<td>setInputMessage</td>
</tr>
<tr>
<td>LOAD</td>
<td>IccProgram</td>
<td>load</td>
</tr>
<tr>
<td>POST</td>
<td>IccClock</td>
<td>setAlarm</td>
</tr>
<tr>
<td>READ</td>
<td>IccFile</td>
<td>readRecord</td>
</tr>
<tr>
<td>READNEXT</td>
<td>IccFileIterator</td>
<td>readNextRecord</td>
</tr>
<tr>
<td>READPREV</td>
<td>IccFileIterator</td>
<td>readPreviousRecord</td>
</tr>
<tr>
<td>READQ TD</td>
<td>IccDataQueue</td>
<td>readItem</td>
</tr>
<tr>
<td>READQ TS</td>
<td>IccTempStore</td>
<td>readItem</td>
</tr>
<tr>
<td>RECEIVE (APPC)</td>
<td>IccSession</td>
<td>receive</td>
</tr>
<tr>
<td>RECEIVE (3270)</td>
<td>IccTerminal</td>
<td>receive, receive3270Data</td>
</tr>
<tr>
<td>RELEASE</td>
<td>IccProgram</td>
<td>unload</td>
</tr>
<tr>
<td>RESETBR</td>
<td>IccFileIterator</td>
<td>reset</td>
</tr>
<tr>
<td>RETRIEVE</td>
<td>IccStartRequestQ</td>
<td>retrieveData</td>
</tr>
<tr>
<td>RETRIEVE INTO, LENGTH</td>
<td>IccStartRequestQ</td>
<td>data</td>
</tr>
<tr>
<td>RETRIEVE QUEUE</td>
<td>IccStartRequestQ</td>
<td>queueName</td>
</tr>
<tr>
<td>RETRIEVE RTRANSID</td>
<td>IccStartRequestQ</td>
<td>returnTransId</td>
</tr>
<tr>
<td>RETRIEVE RTERMD</td>
<td>IccStartRequestQ</td>
<td>returnTermId</td>
</tr>
<tr>
<td>RETURN</td>
<td>IccControl</td>
<td>main</td>
</tr>
<tr>
<td>RETURN TRANSID</td>
<td>IccTerminal</td>
<td>setNextTransId</td>
</tr>
<tr>
<td>RETURN IMMEDIATE</td>
<td>IccTerminal</td>
<td>setNextTransId</td>
</tr>
<tr>
<td>RETURN COMMAREA LENGTH</td>
<td>IccTerminal</td>
<td>setNextCommArea</td>
</tr>
<tr>
<td>RETURN INPUTMSG, INPUTMSGLEN</td>
<td>IccTerminal</td>
<td>setNextInputMessage</td>
</tr>
<tr>
<td>Note: The <strong>retrieveData</strong> method gets the start information from CICS and stores it in the IccStartRequestQ object: the information can then be accessed using <strong>data</strong>, <strong>queueName</strong>, <strong>returnTermid and returnTransId</strong> methods.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note: Returning (using C++ reserved word <strong>return</strong>) from method <strong>run</strong> in class <strong>IccControl</strong> results in an EXEC CICS RETURN.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note: Issue this call before returning from <strong>IccControl::run</strong>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REWRITE</td>
<td>IccFile</td>
<td>rewriteRecord</td>
</tr>
<tr>
<td>SEND (APPC)</td>
<td>IccSession</td>
<td>send, sendInvite, sendLast</td>
</tr>
</tbody>
</table>
### EXEC CICS to Foundation Class methods

<table>
<thead>
<tr>
<th>EXEC CICS</th>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEND (3270)</td>
<td>IccTerminal</td>
<td>send, sendLine</td>
</tr>
<tr>
<td>SEND CONTROL CURSOR</td>
<td>IccTerminal</td>
<td>setCursor setLine, setNewLine</td>
</tr>
<tr>
<td>SEND CONTROL ERASE</td>
<td>IccTerminal</td>
<td>erase</td>
</tr>
<tr>
<td>SEND CONTROL FREEKB</td>
<td>IccTerminal</td>
<td>freeKeyboard</td>
</tr>
<tr>
<td>SET FILE</td>
<td>IccFile</td>
<td>setAccess</td>
</tr>
<tr>
<td>ADD</td>
<td>BROWSE</td>
<td>DELETE</td>
</tr>
<tr>
<td>SET FILE OPEN</td>
<td>IccFile</td>
<td>setStatus</td>
</tr>
<tr>
<td>STATUS</td>
<td>ENABLESTATUS</td>
<td>IccFile</td>
</tr>
<tr>
<td>SIGNOFF</td>
<td>IccTerminal</td>
<td>signoff</td>
</tr>
<tr>
<td>SIGNON</td>
<td>IccTerminal</td>
<td>signon</td>
</tr>
<tr>
<td>START TRANSID AT/AFTER</td>
<td>IccStartRequestQ</td>
<td>start 4</td>
</tr>
<tr>
<td>START TRANSID FROM LENGTH</td>
<td>IccStartRequestQ</td>
<td>setData, registerDataBuffer 4</td>
</tr>
<tr>
<td>START TRANSID NOCHECK</td>
<td>IccStartRequestQ</td>
<td>setStartOpts 4</td>
</tr>
<tr>
<td>START TRANSID PROTECT</td>
<td>IccStartRequestQ</td>
<td>setStartOpts 4</td>
</tr>
<tr>
<td>START TRANSID QUEUE</td>
<td>IccStartRequestQ</td>
<td>setQueueName 4</td>
</tr>
<tr>
<td>START TRANSID REQID</td>
<td>IccStartRequestQ</td>
<td>start 4</td>
</tr>
<tr>
<td>START TRANSID TERMD</td>
<td>IccStartRequestQ</td>
<td>start 4</td>
</tr>
<tr>
<td>START TRANSID USERID</td>
<td>IccStartRequestQ</td>
<td>start 4</td>
</tr>
<tr>
<td>START TRANSID RTERMID</td>
<td>IccStartRequestQ</td>
<td>setReturnTermId 4</td>
</tr>
<tr>
<td>START TRANSID RTRANSID</td>
<td>IccStartRequestQ</td>
<td>setReturnTransId 4</td>
</tr>
</tbody>
</table>

**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                                   |
| SYNCPOINT ROLLLBACK       | 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                     |

Appendix A. Mapping EXEC CICS calls to Foundation Class methods
EXEC CICS to Foundation Class methods
The following table shows the correspondence between CICS calls made using the Foundation Classes and the equivalent EXEC CICS API calls.

<table>
<thead>
<tr>
<th>IccAbendData Class</th>
<th>Method</th>
<th>EXEC CICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>abendCode</td>
<td>ASSIGN ABCODE</td>
<td></td>
</tr>
<tr>
<td>ASRAInterrupt</td>
<td>ASSIGN ASRAINTRPT</td>
<td></td>
</tr>
<tr>
<td>ASRAKeyType</td>
<td>ASSIGN ASRAKEY</td>
<td></td>
</tr>
<tr>
<td>ASRAPSW</td>
<td>ASSIGN ASRAPSW</td>
<td></td>
</tr>
<tr>
<td>ASRARegisters</td>
<td>ASSIGN ASRAREGS</td>
<td></td>
</tr>
<tr>
<td>ASRAStorageType</td>
<td>ASSIGN ASRASPC</td>
<td></td>
</tr>
<tr>
<td>isDumpAvailable</td>
<td>ASSIGN ABDUMP</td>
<td></td>
</tr>
<tr>
<td>originalAbendCode</td>
<td>ASSIGN ORGABCODE</td>
<td></td>
</tr>
<tr>
<td>programName</td>
<td>ASSIGN ABPROGRAM</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IccAbsTime Class</th>
<th>Method</th>
<th>EXEC CICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>FORMATTIME YYDDD/YYMMDD/etc.</td>
<td></td>
</tr>
<tr>
<td>dayOfMonth</td>
<td>FORMATTIME DAYOFMONTH</td>
<td></td>
</tr>
<tr>
<td>dayOfWeek</td>
<td>FORMATTIME DAYOFWEEK</td>
<td></td>
</tr>
<tr>
<td>daysSince1900</td>
<td>FORMATTIME DAYCOUNT</td>
<td></td>
</tr>
<tr>
<td>monthOfYear</td>
<td>FORMATTIME MONTHOFYEAR</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>FORMATTIME TIME</td>
<td></td>
</tr>
<tr>
<td>year</td>
<td>FORMATTIME YEAR</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IccClock Class</th>
<th>Method</th>
<th>EXEC CICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancelAlarm</td>
<td>CANCEL</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>FORMATTIME YYDDD/YYMMDD/etc.</td>
<td></td>
</tr>
<tr>
<td>dayOfMonth</td>
<td>FORMATTIME DAYOFMONTH</td>
<td></td>
</tr>
<tr>
<td>dayOfWeek</td>
<td>FORMATTIME DAYOFWEEK</td>
<td></td>
</tr>
<tr>
<td>daysSince1900</td>
<td>FORMATTIME DAYCOUNT</td>
<td></td>
</tr>
<tr>
<td>monthOfYear</td>
<td>FORMATTIME MONTHOFYEAR</td>
<td></td>
</tr>
<tr>
<td>setAlarm</td>
<td>POST</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>FORMATTIME TIME</td>
<td></td>
</tr>
<tr>
<td>update</td>
<td>ASKTIME</td>
<td></td>
</tr>
<tr>
<td>year</td>
<td>FORMATTIME YEAR</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IccConsole Class</th>
<th>Method</th>
<th>EXEC CICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>write</td>
<td>WRITE OPERATOR</td>
<td></td>
</tr>
</tbody>
</table>
### Foundation Class methods to EXEC CICS

<table>
<thead>
<tr>
<th>Method</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>writeAndGetReply</td>
<td>WRITE OPERATOR</td>
</tr>
<tr>
<td>IccControl Class</td>
<td>EXEC CICS</td>
</tr>
<tr>
<td>callingProgramId</td>
<td>ASSIGN INVOKEPINGPROG</td>
</tr>
<tr>
<td>cancelAbendHandler</td>
<td>HANDLE ABEND CANCEL</td>
</tr>
<tr>
<td>commArea</td>
<td>ADDRESS COMMAREA</td>
</tr>
<tr>
<td>initData</td>
<td>ASSIGN INITPARM &amp; INITPARMLEN</td>
</tr>
<tr>
<td>programId</td>
<td>ASSIGN PROGRAM</td>
</tr>
<tr>
<td>resetAbendHandler</td>
<td>HANDLE ABEND RESET</td>
</tr>
<tr>
<td>setAbendHandler</td>
<td>HANDLE ABEND PROGRAM</td>
</tr>
<tr>
<td>IccDataQueue Class</td>
<td>EXEC CICS</td>
</tr>
<tr>
<td>empty</td>
<td>DELETEQ TD</td>
</tr>
<tr>
<td>readItem</td>
<td>READQ TD</td>
</tr>
<tr>
<td>writeItem</td>
<td>WRITEQ TD</td>
</tr>
<tr>
<td>IccFile Class</td>
<td>EXEC CICS</td>
</tr>
<tr>
<td>access</td>
<td>INQUIRE FILE ADD</td>
</tr>
<tr>
<td>accessMethod</td>
<td>INQUIRE FILE ACCESSMETHOD</td>
</tr>
<tr>
<td>deleteRecord</td>
<td>DELETE FILE ridfld</td>
</tr>
<tr>
<td>deleteLockedRecord</td>
<td>DELETE FILE</td>
</tr>
<tr>
<td>enableStatus</td>
<td>INQUIRE FILE ENABLESTATUS</td>
</tr>
<tr>
<td>isAddable</td>
<td>INQUIRE FILE ADD</td>
</tr>
<tr>
<td>isBrowsable</td>
<td>INQUIRE FILE BROWSE</td>
</tr>
<tr>
<td>isDeleteable</td>
<td>INQUIRE FILE DELETE</td>
</tr>
<tr>
<td>isEmptyOnOpen</td>
<td>INQUIRE FILE EMPTYSTATUS</td>
</tr>
<tr>
<td>isReadable</td>
<td>INQUIRE FILE READ</td>
</tr>
<tr>
<td>isRecoverable</td>
<td>INQUIRE FILE RECOVSTATUS</td>
</tr>
<tr>
<td>isUpdatable</td>
<td>INQUIRE FILE UPDATE</td>
</tr>
<tr>
<td>keyPosition</td>
<td>INQUIRE FILE KEYPOSITION</td>
</tr>
<tr>
<td>openStatus</td>
<td>INQUIRE FILE OPENSTATUS</td>
</tr>
<tr>
<td>readRecord</td>
<td>READ FILE</td>
</tr>
<tr>
<td>recordFormat</td>
<td>INQUIRE FILE RECORDFORMAT</td>
</tr>
<tr>
<td>recordLength</td>
<td>INQUIRE FILE RECORDSIZE</td>
</tr>
<tr>
<td>rewriteRecord</td>
<td>REWRITE FILE</td>
</tr>
<tr>
<td>setAccess</td>
<td>SET FILE ADD BROWSE DELETE etc.</td>
</tr>
<tr>
<td>setEmptyOnOpen</td>
<td>SET FILE EMPTYSTATUS</td>
</tr>
<tr>
<td>setStatus</td>
<td>SET FILE OPENSTATUS ENABLESTATUS</td>
</tr>
<tr>
<td>type</td>
<td>INQUIRE FILE TYPE</td>
</tr>
<tr>
<td>unlockRecord</td>
<td>UNLOCK FILE</td>
</tr>
<tr>
<td>writeRecord</td>
<td>WRITE FILE</td>
</tr>
<tr>
<td>IccFileIterator Class</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>EXEC CICS</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>IccFileIterator (constructor)</td>
<td>STARTBR FILE</td>
</tr>
<tr>
<td>IccFileIterator (destructor)</td>
<td>ENDBR FILE</td>
</tr>
<tr>
<td>readNextRecord</td>
<td>READNEXT FILE</td>
</tr>
<tr>
<td>readPreviousRecord</td>
<td>READPREV FILE</td>
</tr>
<tr>
<td>reset</td>
<td>RESETBR FILE</td>
</tr>
<tr>
<td>IccJournal Class</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>EXEC CICS</td>
</tr>
<tr>
<td>wait</td>
<td>WAIT JOURNALNUM</td>
</tr>
<tr>
<td>writeRecord</td>
<td>WRITE JOURNALNUM</td>
</tr>
<tr>
<td>IccProgram Class</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>EXEC CICS</td>
</tr>
<tr>
<td>link</td>
<td>LINK PROGRAM</td>
</tr>
<tr>
<td>load</td>
<td>LOAD PROGRAM</td>
</tr>
<tr>
<td>unload</td>
<td>RELEASE PROGRAM</td>
</tr>
<tr>
<td>IccResource Class</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>EXEC CICS</td>
</tr>
<tr>
<td>condition</td>
<td>(RESP &amp; RESP2)</td>
</tr>
<tr>
<td>setRouteOption</td>
<td>(SYSID)</td>
</tr>
<tr>
<td>IccSemaphore Class</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>EXEC CICS</td>
</tr>
<tr>
<td>lock</td>
<td>ENQ RESOURCE</td>
</tr>
<tr>
<td>tryLock</td>
<td>ENQ RESOURCE NOSUSPEND</td>
</tr>
<tr>
<td>unlock</td>
<td>DEQ RESOURCE</td>
</tr>
<tr>
<td>IccSession Class</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>EXEC CICS</td>
</tr>
<tr>
<td>allocate</td>
<td>ALLOCATE</td>
</tr>
<tr>
<td>connectProcess</td>
<td>CONNECT PROCESS CONVID</td>
</tr>
<tr>
<td>converse</td>
<td>CONVERSE CONVID</td>
</tr>
<tr>
<td>extractProcess</td>
<td>EXTRACT PROCESS CONVID</td>
</tr>
<tr>
<td>flush</td>
<td>WAIT CONVID</td>
</tr>
<tr>
<td>free</td>
<td>FREE CONVID</td>
</tr>
<tr>
<td>issueAbend</td>
<td>ISSUE ABEND CONVID</td>
</tr>
<tr>
<td>issueConfirmation</td>
<td>ISSUE CONFIRMATION CONVID</td>
</tr>
<tr>
<td>issueError</td>
<td>ISSUE ERROR CONVID</td>
</tr>
<tr>
<td>issuePrepare</td>
<td>ISSUE PREPARE CONVID</td>
</tr>
<tr>
<td>issueSignal</td>
<td>ISSUE SIGNAL CONVID</td>
</tr>
<tr>
<td>receive</td>
<td>RECEIVE CONVID</td>
</tr>
<tr>
<td>send</td>
<td>SEND CONVID</td>
</tr>
<tr>
<td>sendInvite</td>
<td>SEND CONVID INVITE</td>
</tr>
<tr>
<td>sendLast</td>
<td>SEND CONVID LAST</td>
</tr>
<tr>
<td>state</td>
<td>EXTRACT ATTRIBUTES</td>
</tr>
</tbody>
</table>
## Foundation Class methods to EXEC CICS

<table>
<thead>
<tr>
<th>IccStartRequestQ Class</th>
<th>Method</th>
<th>EXEC CICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancel</td>
<td>CANCEL</td>
<td></td>
</tr>
<tr>
<td>retrieveData</td>
<td>RETRIEVE</td>
<td></td>
</tr>
<tr>
<td>start</td>
<td>START TRANSID</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IccSystem Class</th>
<th>Method</th>
<th>EXEC CICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>applName</td>
<td>ASSIGN APPLID</td>
<td></td>
</tr>
<tr>
<td>beginBrowse</td>
<td>INQUIRE (FILE, TDQUEUE, etc) START</td>
<td></td>
</tr>
<tr>
<td>dateFormat</td>
<td>FORMATTIME DATEFORM</td>
<td></td>
</tr>
<tr>
<td>endBrowse</td>
<td>INQUIRE (FILE, TDQUEUE, etc) END</td>
<td></td>
</tr>
<tr>
<td>freeStorage</td>
<td>FREEMAIN</td>
<td></td>
</tr>
<tr>
<td>getFile</td>
<td>INQUIRE FILE</td>
<td></td>
</tr>
<tr>
<td>getNextFile</td>
<td>INQUIRE FILE NEXT</td>
<td></td>
</tr>
<tr>
<td>getStorage</td>
<td>GETMAIN SHARED</td>
<td></td>
</tr>
<tr>
<td>operatingSystem</td>
<td>INQUIRE SYSTEM OPSYS</td>
<td></td>
</tr>
<tr>
<td>operatingSystemLevel</td>
<td>INQUIRE SYSTEM OPREL</td>
<td></td>
</tr>
<tr>
<td>release</td>
<td>INQUIRE SYSTEM RELEASE</td>
<td></td>
</tr>
<tr>
<td>releaseText</td>
<td>INQUIRE SYSTEM RELEASE</td>
<td></td>
</tr>
<tr>
<td>sysId</td>
<td>ASSIGN SYSID</td>
<td></td>
</tr>
<tr>
<td>workArea</td>
<td>ADDRESS CWA</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IccTask Class</th>
<th>Method</th>
<th>EXEC CICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>abend</td>
<td>ABEND</td>
<td></td>
</tr>
<tr>
<td>commitUOW</td>
<td>SYNCPOINT</td>
<td></td>
</tr>
<tr>
<td>delay</td>
<td>DELAY</td>
<td></td>
</tr>
<tr>
<td>dump</td>
<td>DUMP TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>enterTrace</td>
<td>ENTER TRACENUM</td>
<td></td>
</tr>
<tr>
<td>facilityType</td>
<td>ASSIGN STARTCODE, TERMCODE, PRINSYSID, FCI</td>
<td></td>
</tr>
<tr>
<td>freeStorage</td>
<td>FREEMAIN</td>
<td></td>
</tr>
<tr>
<td>isCommandSecurityOn</td>
<td>ASSIGN CMDSEC</td>
<td></td>
</tr>
<tr>
<td>isCommitSupported</td>
<td>ASSIGN STARTCODE</td>
<td></td>
</tr>
<tr>
<td>isResourceSecurityOn</td>
<td>ASSIGN RESSEC</td>
<td></td>
</tr>
<tr>
<td>isRestarted</td>
<td>ASSIGN RESTART</td>
<td></td>
</tr>
<tr>
<td>isStartDataAvailable</td>
<td>ASSIGN STARTCODE</td>
<td></td>
</tr>
<tr>
<td>principalSysId</td>
<td>ASSIGN PRINSYSID</td>
<td></td>
</tr>
<tr>
<td>priority</td>
<td>ASSIGN TASKPRIORITY</td>
<td></td>
</tr>
<tr>
<td>rollBackUOW</td>
<td>SYNCPOINT ROLLBACK</td>
<td></td>
</tr>
<tr>
<td>setPriority</td>
<td>CHANGE TASK PRIORITY</td>
<td></td>
</tr>
<tr>
<td>startType</td>
<td>ASSIGN STARTCODE</td>
<td></td>
</tr>
<tr>
<td>suspend</td>
<td>SUSPEND</td>
<td></td>
</tr>
<tr>
<td>triggerDataQueueId</td>
<td>ASSIGN QNAME</td>
<td></td>
</tr>
</tbody>
</table>
### Foundation Class methods to EXEC CICS

<table>
<thead>
<tr>
<th>Method</th>
<th>EXEC CICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId Assign UserID</td>
<td></td>
</tr>
<tr>
<td>waitExternal Wait EXTERNAL / WAITCICS</td>
<td></td>
</tr>
<tr>
<td>waitOnAlarm Wait EVENT</td>
<td></td>
</tr>
<tr>
<td>workArea ADDRESS TWA</td>
<td></td>
</tr>
<tr>
<td><strong>IccTempStore Class</strong></td>
<td></td>
</tr>
<tr>
<td>empty DELETEQ TS</td>
<td></td>
</tr>
<tr>
<td>readItem READQ TS ITEM</td>
<td></td>
</tr>
<tr>
<td>readNextItem READQ TS NEXT</td>
<td></td>
</tr>
<tr>
<td>rewriteItem WRITEQ TS ITEM REWRITE</td>
<td></td>
</tr>
<tr>
<td>writeItem WRITEQ TS ITEM</td>
<td></td>
</tr>
<tr>
<td><strong>IccTerminal Class</strong></td>
<td></td>
</tr>
<tr>
<td>erase SEND CONTROL ERASE</td>
<td></td>
</tr>
<tr>
<td>freeKeyboard SEND CONTROL FREEKB</td>
<td></td>
</tr>
<tr>
<td>height ASSIGN SCRNHT</td>
<td></td>
</tr>
<tr>
<td>netName ASSIGN NETNAME</td>
<td></td>
</tr>
<tr>
<td>receive RECEIVE</td>
<td></td>
</tr>
<tr>
<td>receive3270Data RECEIVE BUFFER</td>
<td></td>
</tr>
<tr>
<td>send SEND</td>
<td></td>
</tr>
<tr>
<td>sendLine SEND</td>
<td></td>
</tr>
<tr>
<td>setCursor SEND CONTROL CURSOR</td>
<td></td>
</tr>
<tr>
<td>setLine SEND CONTROL CURSOR</td>
<td></td>
</tr>
<tr>
<td>setNewLine SEND CONTROL CURSOR</td>
<td></td>
</tr>
<tr>
<td>signoff SIGNOFF</td>
<td></td>
</tr>
<tr>
<td>signon SIGNON</td>
<td></td>
</tr>
<tr>
<td>waitForAID RECEIVE</td>
<td></td>
</tr>
<tr>
<td>width ASSIGN SCRNWD</td>
<td></td>
</tr>
<tr>
<td>workArea ADDRESS TCTUA</td>
<td></td>
</tr>
<tr>
<td><strong>IccTerminalData Class</strong></td>
<td></td>
</tr>
<tr>
<td>alternateHeight ASSIGN ALTSCRNHT</td>
<td></td>
</tr>
<tr>
<td>alternateWidth ASSIGN ALTSCRNWD</td>
<td></td>
</tr>
<tr>
<td>defaultHeight ASSIGN DEFSCRNHT</td>
<td></td>
</tr>
<tr>
<td>defaultWidth ASSIGN DEFSCRNWD</td>
<td></td>
</tr>
<tr>
<td>graphicCharSetId ASSIGN GCHARS</td>
<td></td>
</tr>
<tr>
<td>graphicCharCodeSet ASSIGN GCODES</td>
<td></td>
</tr>
<tr>
<td>isAPLKeyboard ASSIGN APLKYBD</td>
<td></td>
</tr>
<tr>
<td>isAPLText ASSIGN APLTEXT</td>
<td></td>
</tr>
<tr>
<td>isBTrans ASSIGN BTRANS</td>
<td></td>
</tr>
<tr>
<td>isColor ASSIGN COLOR</td>
<td></td>
</tr>
<tr>
<td>isEWA ASSIGN ESASUPP</td>
<td></td>
</tr>
</tbody>
</table>

Appendix B. Mapping Foundation Class methods to EXEC CICS calls
### Foundation Class methods to EXEC CICS

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isExtended3270</td>
<td>ASSIGN EXTDS</td>
</tr>
<tr>
<td>isGoodMorning</td>
<td>ASSIGN GMMI</td>
</tr>
<tr>
<td>isHighlight</td>
<td>ASSIGN HILIGHT</td>
</tr>
<tr>
<td>isKatakana</td>
<td>ASSIGN KATAKANA</td>
</tr>
<tr>
<td>isMSRControl</td>
<td>ASSIGN MSRCONTROL</td>
</tr>
<tr>
<td>isFieldOutline</td>
<td>ASSIGN OUTLINE</td>
</tr>
<tr>
<td>isPS</td>
<td>ASSIGN PS</td>
</tr>
<tr>
<td>isSOSI</td>
<td>ASSIGN SOSI</td>
</tr>
<tr>
<td>isTextKeyboard</td>
<td>ASSIGN TEXTKYBD</td>
</tr>
<tr>
<td>isTextPrint</td>
<td>ASSIGN TEXTPRINT</td>
</tr>
<tr>
<td>isValidation</td>
<td>ASSIGN VALIDATION</td>
</tr>
</tbody>
</table>

#### IccUser Class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>changePassword</td>
<td>CHANGE PASSWORD</td>
</tr>
<tr>
<td>verifyPassword</td>
<td>VERIFY PASSWORD</td>
</tr>
</tbody>
</table>
Appendix C. Output from sample programs

This section shows the typical screen output from the supplied sample programs (see "Sample source code" on page 6).

**ICC$BUF (IBUF)**

This is program 'icc$buf'...

<table>
<thead>
<tr>
<th>IccBuf buf1</th>
<th>dal= 0 dl= 0 E+I</th>
</tr>
</thead>
<tbody>
<tr>
<td>IccBuf buf2(50)</td>
<td>dal=50 dl= 0 E+I</td>
</tr>
<tr>
<td>IccBuf buf3(30,fixed)</td>
<td>dal=30 dl= 0 F+I</td>
</tr>
<tr>
<td>IccBuf buf4(sizeof(AStruct),&amp;aStruc)</td>
<td>dal=24 dl=24 F+E [Some text for aStruc]</td>
</tr>
<tr>
<td>IccBuf buf5(&quot;A String Literal&quot;)</td>
<td>dal=19 dl=19 E+I [Some data somewhere]</td>
</tr>
<tr>
<td>IccBuf buf6(buf5)</td>
<td>dal=19 dl=19 E+I [Some data somewhere]</td>
</tr>
<tr>
<td>buf1 = &quot;Some XXX data for buf1&quot;</td>
<td>dal=22 dl=22 E+I [Some XXX data for buf1]</td>
</tr>
<tr>
<td>buf2.assign(strlen(data),data)</td>
<td>dal=50 dl=19 E+I [Some data somewhere]</td>
</tr>
<tr>
<td>buf1.cut(4,5)</td>
<td>dal=22 dl=18 E+I [Some data for buf1]</td>
</tr>
<tr>
<td>buf5.insert(5,more,5)</td>
<td>dal=24 dl=24 E+I [Some more data somewhere]</td>
</tr>
<tr>
<td>buf5.replace4,xtra,5</td>
<td>dal=24 dl=24 E+I [Some xtra data somewhere]</td>
</tr>
<tr>
<td>buf2 &lt;&lt; &quot;.ext&quot;</td>
<td>dal=50 dl=23 E+I [Some data somewhere.ext]</td>
</tr>
<tr>
<td>buf3 = buf4</td>
<td>dal=30 dl=24 F+I [Some text for aStruc]</td>
</tr>
</tbody>
</table>

(buf3 == buf4) returns true (OK).

(buf3 = "garbage") dal=30 dl= 7 F+I [garbage]

(buf3 != buf4) returns true (OK).

Program 'icc$buf' complete: Hit PF12 to End

**ICC$CLK (ICLK)**

This is program 'icc$clk' ...

| date() = [220296 ] |
| date(DOMMY) = [220296 ] |
| date(DOMMY,':') = [22:02:96] |
| date(MDYY) = [022296 ] |
| date(YMD) = [96053 ] |
| daysSince1900() = 35116 |
| dayOfWeek() = 4 Today is NOT Friday |
| dayOfMonth() = 22 |
| monthOfYear() = 2 |
| time() = [143832 ] |
| time('-') = [14-38-32] |
| year() = [1996] |

Program 'icc$clk' complete: Hit PF12 to End

**ICC$DAT (IDAT)**

This is program 'icc$dat'...

Writing records to 'ICCQ'...

- writing record 1: 'Hello World' - item 1' <NORMAL>
- writing record 2: 'Hello World' - item 2' <NORMAL>
- writing record 3: 'Hello World' - item 3' <NORMAL>

Reading records back in...

- reading record 1: 'Hello World' - item 1' <NORMAL>
- reading record 2: 'Hello World' - item 2' <NORMAL>
- reading record 3: 'Hello World' - item 3' <NORMAL>

Program 'icc$dat' complete: Hit PF12 to End
Output from sample programs

**ICC$EXC1 (IEX1)**

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

**ICC$EXC2 (IEX2)**

This is program 'icc$exc2'...
Creating IccTermId id1...
Creating IccTermId id2...
IccException: 112 IccTermId::IccTermId type=invalidArgument (IccMessage: 030 IccTermId::IccTermId <Invalid string length passed to 'IccTermId' constructor.
Specified: 5, Maximum allowed: 4>)
Program 'icc$exc2' complete: Hit PF12 to End

**ICC$EXC3 (IEX3)**

This is program 'icc$exc3'...
About to read Temporary Storage 'UNKNOWN!'...
IccException: 094 IccTempStore::readNextItem type=CICSCondition (IccMessage: 008 IccTempStore::readNextItem <CICS returned the 'QIDERR' condition.>)
Program 'icc$exc3' complete: Hit PF12 to End
**ICC$FIL (IFIL)**

This is program 'icc$fil'...
Deleting records in file 'ICCKFILE'...
5 records were deleted.
Writing records to file 'ICCKFILE'...
- writing record number 1.  <NORMAL>
- writing record number 2.  <NORMAL>
- writing record number 3.  <NORMAL>
- writing record number 4.  <NORMAL>
- writing record number 5.  <NORMAL>
Browsing records...
- record read: [BACH, J S 003 00-1234 BACH ]
- record read: [CHOPIN, F 004 00-3355 CHOPIN ]
- record read: [HANDEL, G F 005 00-4466 HANDEL ]
- record read: [BEETHOVEN, L 007 00-2244 BEET ]
- record read: [MOZART, W A 008 00-5577 WOLFGANG ]
- record read: [MOZART, W A 008 00-5577 WOLFGANG ]
- record read: [BEETHOVEN, L 007 00-2244 BEET ]
- record read: [HANDEL, G F 005 00-4466 HANDEL ]
- record read: [CHOPIN, F 004 00-3355 CHOPIN ]
- record read: [BACH, J S 003 00-1234 BACH ]
Updating record 1...
readRecord(update)<NORMAL> rewriteRecord()<NORMAL>
- record read: [MOZART, W A 008 00-5678 WOLFGANG ]
Program 'icc$fil' complete: Hit PF12 to End

**ICC$HEL (IHEL)**

Hello World

**ICC$JRN (IJRN)**

This is program 'icc$jrn'...
Writing 3 records to journal number 77...
- writing record 1: [Hello World - item 1]  <NORMAL>
- writing record 2: [Hello World - item 2]  <NORMAL>
- writing record 3: [Hello World - item 3]  <NORMAL>
Program 'icc$jrn' complete: Hit PF12 to End
Output from sample programs

**ICC$PRG1 (IPR1)**

**First Screen**

This is program 'icc$prg1'

Loaded program: ICC$PRG2 <NORMAL> Length=0 Address=ff000000
Unloading program: ICC$PRG2 <NORMAL>
- Hit ENTER to continue...

**Second Screen**

About to link to program 'ICC$PRG2'
- commArea before link is [DATA SET BY ICC$PRG1]
- Hit ENTER to continue...
  - This is program 'icc$prg2'
  - commArea received from caller = [DATA SET BY ICC$PRG1]
  - Changed commArea to [DATA RETURNED BY ICC$PRG2]
  - Hit ENTER to return to caller...
  - link call returned <NORMAL>
  - commArea after link is [DATA RETURNED BY ICC$PRG2]

About to link to program 'ICC$PRG3' on system 'ICC2'
- commArea before link is [DATA SET BY ICC$PRG1]
- Hit ENTER to continue...
  - link call returned <NORMAL>
  - commArea after link is [DATA RETURNED BY ICC$PRG3]

Program 'icc$prg1' complete: Hit PF12 to End

**ICC$RES1 (IRE1)**

This is program 'icc$res1'

Writing items to CustomDataQueue 'ICCQ'...
- writing item #1: 'Hello World - item 1' <NORMAL>
- writing item #2: 'Hello World - item 2' <NORMAL>
- writing item #3: 'Hello World - item 3' <NORMAL>

Reading items from CustomDataQueue 'ICCQ'...
- item = 'Hello World - item 1'
- item = 'Hello World - item 2'
- item = 'Hello World - item 3'

Reading loop complete.
> In handleEvent().
Summary=IccEvent: CustomDataQueue::readItem condition=23 (QZ ERO) minor=0
Program 'icc$res1' complete: Hit PF12 to End
Output from sample programs

ICCSRES2 (IRE2)

This is program 'iccsres2'...
invoking clear() method for IccDataQueue object <NORMAL>
invoking clear() method for IccTempStore object <NORMAL>
put() item #1 in IccDataQueue object
put() item #2 in IccDataQueue object
put() item #3 in IccDataQueue object
put() item #1 in IccTempStore object
put() item #2 in IccTempStore object
put() item #3 in IccTempStore object
Now get items from IccDataQueue object
get() from IccDataQueue object returned 'Hello World - item 1'
get() from IccDataQueue object returned 'Hello World - item 2'
get() from IccDataQueue object returned 'Hello World - item 3'
get() from IccTempStore object returned 'Hello World - item 1'
get() from IccTempStore object returned 'Hello World - item 2'
get() from IccTempStore object returned 'Hello World - item 3'
Program 'iccsres2' complete: Hit PF12 to End

ICCSSEM (ISEM)

This is program 'iccssem'...
Constructing IccSemaphore object (lock by value)... <NORMAL>
Issuing lock request... <NORMAL>
Constructing Semaphore object (lock by address)... <NORMAL>
Issuing tryLock request... <NORMAL>
Issuing unlock request... <NORMAL>
Program 'iccssem' complete: Hit PF12 to End

ICCSSES1 (ISE1)

This is program 'iccses1'...
allocate session... <NORMAL>
STATE=81 ALLOCATED ERR=0 connectProcess... <NORMAL>
STATE=90 SEND ERR=0 sendInvite... <NORMAL>
STATE=87 PENDRECEIVE ERR=0 receive... <NORMAL>
STATE=85 FREE ERR=0 - data from back end=[Hi there this is from backEnd
TIME=14:49:18 on 22/02/96]
free... <NORMAL>
STATE=1 NOTAPPLIC ERR=0
Program 'iccses1' complete: Hit PF12 to End
Output from sample programs

**ICC$SES2 (ISE2)**

This screen is typical output after running “CEBR DTPBKEND” on the back-end CICS system:

```
CEBR TSQ DTPBKEND SYSID ABCD REC 1 OF 11 COL 1 OF 78
ENTER COMMAND ===>  
                       ********************** TOP OF QUEUE **********************
00001 Transaction 'ISE2' starting.
00002 extractProcess...
00003 <NORMAL> STATE=88 RECEIVE ERR=0
00004 process=[ISE2] syncLevel=1 PIP=[Hello World]
00005 receive...
00006 <NORMAL> STATE=90 SEND ERR=0 NoData=0
00007 data from front end=[Hi there this is from frontEnd TIME=16:03:18 on 04/0
00008 sendLast ...
00009 <NORMAL> STATE=86 PENDFREE ERR=0
00010 free...
00011 <NORMAL> STATE=1 NOTAPPLIC ERR=0
                       ********************** BOTTOM OF QUEUE **********************
PF1 : HELP PF2 : SWITCH HEX/CHAR PF3 : TERMINATE BROWSE
PF4 : VIEW TOP PF5 : VIEW BOTTOM PF6 : REPEAT LAST FIND
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : VIEW RIGHT
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: UNDEFINED
```

**ICC$SRQ1 (ISR1)**

This is program 'icc$srq1'...
Starting Tran 'ISR2' on terminal 'PE12' after 5 seconds... - <NORMAL>
request='DFIU0000'
Issuing cancel for start request='DFIU0000'... - <NORMAL>
request='DFIU0000'
Starting Tran 'ISR2' on terminal 'PE12' after 5 seconds... - <NORMAL>
request='REQUEST1'
Program 'icc$srq1' complete.

**ICC$SRQ2 (ISR2)**

This is program 'icc$srq2'...
retrieveData()... - <NORMAL>
Start buffer contents = [This is a greeting from program 'icc$srq1'!!]
Start queue= [startqnm]
Start rtm= [ITMP]
Start rtm= [PE11]
Sleeping for 5 seconds...
Starting tran 'ITMP' on terminal 'PE11' on system ICC1...<NORMAL>
Program 'icc$srq2' complete: Hit PF12 to end
ICC$SYS (ISYS)

This is program 'icc$sys'...
applName=ICC$REG01 operatingSystem=A operatingSystemLevel=41
releaseText=[0210] sysidnt=ICC1
getStorage( 5678, 'Y')...  <NORMAL>
freeStorage( p )...  <NORMAL>
Checking attributes of a named file (ICKFILE)...
>ICCKFILE< Add=true Brw=true Del=true Read=true Upd=true op=18 en=23
accessMethod=3 isRecoverable=true keyLength=16 keyPosition=16
setStatus( closed ) ...  <NORMAL>
setStatus( disabled ) ...  <NORMAL>
setStatus( notUpdatable ) ...  <NORMAL>
>ICCKFILE< Add=true Brw=true Del=true Read=true Upd=false op=19 en=24
setAccess( updateable ) & setStatus( enabled, open ) ...
>ICCKFILE< Add=true Brw=true Del=true Read=true Upd=true op=18 en=23
Beginning browse of all file objects in CICS system...  <NORMAL>
- >ICCEFILE< type=1  <NORMAL>
- >ICCKFILE< type=6  <NORMAL>
- >ICCRFILE< type=1  <NORMAL>
Program 'icc$sys' complete: Hit PF12 to End

ICC$TMP (ITMP)

This is program 'icc$tmp'...
Writing 3 records to IccTempStore object 'ICCSTORE'...
- writing record #1: 'Hello World - item 1'  <NORMAL>
- writing record #2: 'Hello World - item 2'  <NORMAL>
- writing record #3: 'Hello World - item 3'  <NORMAL>
Reading records back in & rewriting new buffer contents...
- record #1 = [Hello World - item 1] - rewriteItem #1  <NORMAL>
- record #2 = [Hello World - item 2] - rewriteItem #2  <NORMAL>
- record #3 = [Hello World - item 3] - rewriteItem #3  <NORMAL>
Reading records back in one last time...
- record #1 = [Modified Hello World - item 1]
- record #1 = [Modified Hello World - item 2]
- record #1 = [Modified Hello World - item 3]
Program 'icc$tmp' complete: Hit PF12 to end

ICC$TRM (ITRM)

This is program 'icc$trm'...
First part of the line...... a continuation of the line.
Start this on the next line  Send this to col 40 of current line
Send this to row 5, column 10  Send this to row 6, column 40
A Red line!
A Blue, reverse video line!
A cout style interface...
you can chain input together; use different types, eg numbers: 123 4567890 12345
6.789123
... and everything is buffered till you issue a flush.
Program 'icc$trm' complete: Hit PF12 to End
This is program 'icc$tsk'...
startType() = terminalInput
number() = 0598
isStartDataSupplied() = true
isCommitSupported() = true
userId() = [rabcics]
enterTrace(77, "ICCENTRY", buffer) <NORMAL>
suspend()... <NORMAL>
delay(ti) (for 2 seconds)... <NORMAL>
getStorage(1234, 'X')... <NORMAL>
freeStorage(p)... <NORMAL>
commitUOW()... <NORMAL>
rollBackUOW()... <NORMAL>

Program 'icc$tsk' complete: Hit PF12 to End OR PF24 to ABEND
Glossary

abstract class. A class that is used as a base class for other classes and has at least one pure virtual function. It is not possible to create an instance of this class.

base class. A class from which other classes are derived.

CICS program. A program that runs in the CICS environment as part of a transaction.

class. A group of objects that share a common definition and common properties, operations and behavior.

class definition. How a class is defined in C++.

class implementation. How a class is implemented in C++.

const. In C++, the const attribute explicitly declares a data object as a data item that cannot be changed. Its value is set at initialization.

constructor. In C++, a special class member function (method) that has the same name as the class and is used to initialize class objects.

default argument. In C++, a default is used when an argument in a method call is not explicitly provided.

delete. A C++ operator that deallocates dynamic storage to destroy an object.

destructor. In C++, a special class member function (method) that has the same name as the class, preceded by (tilde), and is executed when an object is destroyed.

distributed program link. A technique where a program running on one CICS system links to a program running on another system.

encapsulation. The means whereby the inner workings of an object are hidden. An application programmer only has direct access to the external features.

function shipping. A technique whereby a transaction running on one CICS system accesses resources held on another system.

inheritance. The passing of class resources or attributes from a base class to a subclass.

method. An operator or function that is declared as a member of a class.

new. A C++ operator that allocates dynamic storage to create an object.

object. An abstraction consisting of data and the operations associated with that data.

overloading. The redefinition of functions and most standard C++ operators. This typically extends the operations that the function or operator performs to different data types.

polymorphism. The application of a method or function to objects of more than one data type.

subclass. A class that is derived from another class. The subclass inherits the data and methods of the base class and can define new methods or over-ride existing methods to define new behavior not inherited from the parent class.

task. One instance of the execution of a particular CICS transaction.

transaction. One or more programs on a CICS server that can be initiated on request by a CICS user.

transaction routing. A technique whereby a transaction initiated on one CICS system is actually run on another system.

UOW. A CICS unit of work is a set of resource updates.

virtual function. In C++, a class member function that is defined with the keyword virtual. The code that is executed when you make a call to a virtual function depends on the type of object for which it is called.
Bibliography

CICS Transaction Server for z/OS

The above titles are the only unlicensed books available in hardcopy for CICS Transaction Server for z/OS Version 2 Release 1. All the remaining CICS and CICSPlex SM books are supplied in softcopy only in the CICS Information Center, which is distributed on CD-ROM.

CICS books for CICS Transaction Server for z/OS

General
- CICS User's Handbook
- CICS Transaction Server for z/OS Glossary

Administration
- CICS System Definition Guide
- CICS Customization Guide
- CICS Resource Definition Guide
- CICS Operations and Utilities Guide
- CICS Supplied Transactions

Programming
- CICS Application Programming Guide
- CICS Application Programming Reference
- CICS System Programming Reference
- CICS Front End Programming Interface User's Guide
- CICS C++ O0 Class Libraries
- CICS Distributed Transaction Programming Guide
- CICS Business Transaction Services
- Java Applications in CICS

Diagnosis
- CICS Problem Determination Guide
- CICS Messages and Codes
- CICS Diagnosis Reference
- CICS Data Areas
- CICS Trace Entries
- CICS Supplementary Data Areas

Communication
- CICS Intercommunication Guide
- CICS Family: Interproduct Communication
- CICS Family: Communicating from CICS on System/390
- CICS External Interfaces Guide
- CICS Internet Guide

Special topics
- CICS Recovery and Restart Guide
- CICS Performance Guide
- CICS IMS Database Control Guide
- CICS RACF Security Guide

© Copyright IBM Corp. 1989, 2001
CICSPlex SM books for CICS Transaction Server for z/OS

General
- CICSPlex SM Concepts and Planning
- CICSPlex SM User Interface Guide
- CICSPlex SM Commands Reference Summary
- CICSPlex SM Web User Interface Guide

Administration and Management
- CICSPlex SM Administration
- CICSPlex SM Operations Views Reference
- CICSPlex SM Monitor Views Reference
- CICSPlex SM Managing Workloads
- CICSPlex SM Managing Resource Usage
- CICSPlex SM Managing Business Applications

Programming
- CICSPlex SM Application Programming Guide
- CICSPlex SM Application Programming Reference

Diagnosis
- CICSPlex SM Resource Tables Reference
- CICSPlex SM Messages and Codes
- CICSPlex SM Problem Determination

Other CICS books
- Designing and Programming CICS Applications
- CICS Application Migration Aid Guide
- CICS Family: API Structure
- CICS Family: Client/Server Programming
- CICS Transaction Gateway for OS/390 Administration
- CICS Family: General Information
- CICS 4.1 Sample Applications Guide
- CICS/ESA 3.3 XRF Guide

Note: The CICS Transaction Server for OS/390: Planning for Installation book that was part of the library for CICS Transaction Server for OS/390, Version 1 Release 3, is now merged with the CICS Transaction Server for z/OS Installation Guide. If you have any questions about the CICS Transaction Server for z/OS library, see CICS Transaction Server for z/OS Installation Guide which discusses both hardcopy and softcopy books and the ways that the books can be ordered.

Related books
Here are some more books that you may find useful.

C++ Programming
You should read the books supplied with your C++ compiler.
The following are some non-IBM publications that are generally available. This is not an exhaustive list. IBM does not specifically recommend these books, and other publications may be available in your local library or bookstore.


### CICS client manuals

<table>
<thead>
<tr>
<th>CICS Clients: Administration</th>
<th>SC33-1792</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS Clients: Messages</td>
<td>SC33-1793</td>
</tr>
<tr>
<td>CICS Clients: Gateways</td>
<td>SC33-1821</td>
</tr>
<tr>
<td>CICS Family: OO Programming in C++ for CICS Clients</td>
<td>SC33-1923</td>
</tr>
<tr>
<td>CICS Family: OO Programming in BASIC for CICS Clients</td>
<td>SC33-1924</td>
</tr>
</tbody>
</table>

### Determining if a publication is current

IBM regularly updates its publications with new and changed information. When first published, both hardcopy and BookManager® softcopy versions of a publication are usually in step. However, due to the time required to print and distribute hardcopy books, the BookManager version is more likely to have had last-minute changes made to it before publication.

Subsequent updates will probably be available in softcopy before they are available in hardcopy. This means that at any time from the availability of a release, softcopy versions should be regarded as the most up-to-date.

For CICS Transaction Server books, these softcopy updates appear regularly on the *Transaction Processing and Data Collection Kit* CD-ROM, SK2T-0730-xx. Each reissue of the collection kit is indicated by an updated order number suffix (the -xx part). For example, collection kit SK2T-0730-06 is more up-to-date than SK2T-0730-05. The collection kit is also clearly dated on the cover.

Updates to the softcopy are clearly marked by revision codes (usually a “#” character) to the left of the changes.
Index

Special Characters
... (parameter)  
in sendLine  231

Numerics
0 (zero)  
in actionOnConditionAsChar  169

A
A  
in actionOnConditionAsChar  169  
in operatingSystem  203
abend  
in IccTask class  207  
in Parameter level  53
abend codes  47
abendCode  
in IccAbendData class  67
abendCode (parameter)  
in abend  207
abendData  
in IccTask class  207
AbendDumpOpt  
in Enumerations  214  
in IccTask class  214
AbendHandlerOpt  
in Enumerations  214  
in IccTask class  214
abendTask  
in ActionOnCondition  174  
in CICS conditions  50
absTime  
in IccClock class  91  
in Type  244
absTime (parameter)  
in Constructor  73  
in operator=  74
access  
in IccFile class  121
Access  
in Enumerations  129  
in IccFile class  129
access (parameter)  
in setAccess  127
Accessing start data  
in Starting transactions asynchronously  34  
in Using CICS Services  34
accessMethod  
in IccFile class  122
action (parameter)  
in setActionOnAnyCondition  172  
in setActionOnCondition  172
actionOnCondition  
in IccResource class  169
ActionOnCondition  
in Enumerations  173  
in IccResource class  173
actionOnConditionAsChar  
in IccResource class  169
actions (parameter)  
in setActionsOnConditions  172
actionsOnConditionsText  
in IccResource class  170
Activating the trace output  
in Debugging Programs  46  
in Tracing a Foundation Class Program  46
addable  
in Access  129
address  
in IccProgram class  157
AID  
in IccTerminal class  225
aid (parameter)  
in waitForAID  234
AIDVal  
in Enumerations  235  
in IccTerminal class  235
AIX, CICS for  
in Platform differences  52
allocate  
in IccSession class  184
AllocateOpt  
in Enumerations  191  
in IccSession class  191
alternateHeight  
in IccTerminalData class  237  
in Public methods  237
alternateWidth  
in IccTerminalData class  237  
in Public methods  237
append  
in IccBuf class  84
applName  
in IccSystem class  201
ASRAInterrupt  
in IccAbendData class  67  
in Public methods  67
ASRAKeyType  
in IccAbendData class  68
in IccAbendData class  68  
in Public methods  68
ASRAStorageType  
in IccAbendData class  68
ASRASpaceType  
in IccAbendData class  69
in Public methods  69
ASRASpaceType  
in IccAbendData class  69
in Public methods  69
buffer (parameter)

- in Constructor 84
- in operator= 86
- in operator!= 87
- in operator+= 87
- in operator== 87
- in operator<< 87
- in put 110, 140, 171, 218
- in registerData 194
- in rewriteRecord 126
- in send 229
- in send3270 230
- in sendLine 230, 231
- in writeRecord 128

Buffer objects
- Data area extensibility 23
- Data area ownership 23
- IccBuf constructors 23
- IccBuf methods 24
- Working with IccResource subclasses 24
- buffers 23, 25
- byAddress
  - in LockType 181
- byValue
  - in LockType 181

C

C++ exceptions 47

C++ Exceptions and the Foundation Classes
- in Conditions, errors, and exceptions 47

callHandleEvent
- in ActionOnCondition 174
- in CICS conditions 50

calling conventions 58

Calling methods on a resource object
- in Overview of the foundation classes 20
- in Using CICS resources 20

callingProgramId
- in IccControl class 103
- in Public methods 103

cancel
- in Cancelling unexpired start requests 34
- in IccRequestld class 167
- in IccStartRequestQ class 193

cancelAbendHandler
- in IccControl class 103

cancelAlarm
- in IccClock class 91

Cancelling unexpired start requests
- in Starting transactions asynchronously 34
- in Using CICS Services 34

Case
- in Enumerations 235
- in IccTerminal class 235

caseOpt (parameter)
- in receive 228
- in receive3270Data 229

Below
- in StorageOpts 215

Blink
- in Highlight 235

Blue
- in Color 235

Bool
- in Enumerations 65
- in Icc structure 65

BoolSet
- in Enumerations 65
- in Icc structure 65

boolText
- in Functions 63
- in Icc structure 63

Browsable
- in Access 129

browsing records 30

Browsing records
- in File control 30
- in Using CICS Services 30

Buf (parameter)
- in dump 208
- in put 228
- in send3270 230
- in sendLine 231
- in setData 195

buffer
- in Example of starting transactions 35, 36

AuxStorage
- in Location 220

BeginBrowse
- in IccSystem class 201

BeginInsert
- in Writing records 28
- in IccFile class 122
- in Public methods 122

Automatic condition handling (callHandleEvent)
- in CICS conditions 50
- in Conditions, errors, and exceptions 50

Automatic creation 13

Automatic deletion 13

Assign
- in Example of file control 31
- in IccBuf class 84, 85
- in IccKey class 147

Automatic
- in UpdateMode 95

Automatic condition handling (callHandleEvent)
- in CICS conditions 50
- in Conditions, errors, and exceptions 50

Automatic creation 13

Automatic deletion 13

AuxStorage
- in Location 220

Base class
- overview 15

Base classes
- in Overview of the foundation classes 15

BaseName (parameter)
- in NameOpt 81

BASESPACE
- in ASRASpaceType 69

BDAM 27

BeginBrowse
- in IccSystem class 201

BeginInsert
- in Writing records 28
- BeginInsert(VSAM only)
  - in IccFile class 122
  - in Public methods 122

Below
- in StorageOpts 215

Blink
- in Highlight 235

Blue
- in Color 235

Bool
- in Enumerations 65
- in Icc structure 65

BoolSet
- in Enumerations 65
- in Icc structure 65

boolText
- in Functions 63
- in Icc structure 63

Browsable
- in Access 129

browsing records 30

Browsing records
- in File control 30
- in Using CICS Services 30

Buf (parameter)
- in dump 208
- in put 228
- in send3270 230
- in sendLine 231
- in setData 195

buffer
- in Example of starting transactions 35, 36

buffer (parameter)
- in Constructor 84
- in operator= 86
- in operator!= 87
- in operator+= 87
- in operator== 87
- in operator<< 87
- in put 110, 140, 171, 218
- in registerData 194
- in rewriteRecord 126
- in send 229
- in send3270 230
- in sendLine 230, 231
- in writeRecord 128

Buffer objects
- Data area extensibility 23
- Data area ownership 23
- IccBuf constructors 23
- IccBuf methods 24
- Working with IccResource subclasses 24
- buffers 23, 25
- byAddress
  - in LockType 181
- byValue
  - in LockType 181

C

C++ exceptions 47

C++ Exceptions and the Foundation Classes
- in Conditions, errors, and exceptions 47

callHandleEvent
- in ActionOnCondition 174
- in CICS conditions 50

calling conventions 58

Calling methods on a resource object
- in Overview of the foundation classes 20
- in Using CICS resources 20

callingProgramId
- in IccControl class 103
- in Public methods 103

cancel
- in Cancelling unexpired start requests 34
- in IccRequestld class 167
- in IccStartRequestQ class 193

cancelAbendHandler
- in IccControl class 103

cancelAlarm
- in IccClock class 91

Cancelling unexpired start requests
- in Starting transactions asynchronously 34
- in Using CICS Services 34

Case
- in Enumerations 235
- in IccTerminal class 235

caseOpt (parameter)
- in receive 228
- in receive3270Data 229

assign
- in Example of file control 31
- in IccBuf class 84, 85
- in IccKey class 147

automatic
- in UpdateMode 95

Automatic condition handling (callHandleEvent)
- in CICS conditions 50
- in Conditions, errors, and exceptions 50

automatic creation 13

automatic deletion 13

auxStorage
- in Location 220

B

base class
- overview 15

Base classes
- in Overview of the foundation classes 15

baseName (parameter)
- in NameOpt 81

BASESPACE
- in ASRASpaceType 69

BDAM 27

beginBrowse
- in IccSystem class 201

beginInsert
- in Writing records 28
- BeginInsert(VSAM only)
  - in IccFile class 122
  - in Public methods 122

Below
- in StorageOpts 215

Blink
- in Highlight 235

Blue
- in Color 235

Bool
- in Enumerations 65
- in Icc structure 65

BoolSet
- in Enumerations 65
- in Icc structure 65

boolText
- in Functions 63
- in Icc structure 63

Browsable
- in Access 129

browsing records 30

Browsing records
- in File control 30
- in Using CICS Services 30

Buf (parameter)
- in dump 208
- in put 228
- in send3270 230
- in sendLine 231
- in setData 195

buffer
- in Example of starting transactions 35, 36
catch
  in C++ Exceptions and the Foundation Classes 47, 48
  in Exception handling (throwException) 51, 52
  in main function 266

CATCH
  in Functions 63
  in Icc structure 63

CEDF (CICS Execution Diagnostic Facility) 46

ch (parameter)
  in operator<< 87, 88, 227

changePassword
  in IccUser class 255
  in Public methods 255

char
  in C++ Exceptions and the Foundation Classes 48

checkOpt
  in Enumerations 198
  in IccStartRequestQ class 198

CICS
  in ASRAStorageType 69
  in GetOpt 66

CICS conditions
  abendTask 52
  automatic condition handling 50
  Automatic condition handling (callHandleEvent) 50
  callHandleEvent 50
  exception handling 51
  Exception handling (throwException) 51
  in Conditions, errors, and exceptions 49
  manual condition handling 50
  Manual condition handling (noAction) 50
  noAction 50
  severe error handling 52
  Severe error handling (abendTask) 52
  throwException 51

CICS Execution Diagnostic Facility (CEDF) 46

CICS for AIX
  in Platform differences 52

CICS OS/2
  in Platform differences 52

CICS resources 19

CICSCondition
  in C++ Exceptions and the Foundation Classes 49
  in Type 119

CICSDataKey
  in StorageOpts 215

CICSEXECKEY
  in ASRAKeyType 68

CICSInternalTask
  in StartType 215

CICSTS13.CICS.SDFHSAMP 6
CICSTS21.CICS.SDFHC370 6
CICSTS21.CICS.SDFHLOAD 7
CICSTS21.CICS.SDFHPROC 6
CICSTS21.CICS.SDFHSAMP 6
CICSTS21.CICS.SDFHSDCK 6

class
  base 15
  resource 17
  resource identification 16

class (continued)
  singleton 20
  support 18

ClassMemoryMgmt
  in Enumerations 65
  in Icc structure 65

className
  in IccBase class 79
  in IccEvent class 115
  in IccException class 118
  in IccMessage class 153

className (parameter)
  in Constructor 117, 153
  in setClassName 80

classType
  in IccBase class 79
  in IccEvent class 115
  in IccException class 118

ClassType
  in Enumerations 80
  in IccBase class 80

classType (parameter)
  in Constructor 117, 169

clear
  in Example of polymorphic behavior 57
  in IccDataQueue class 109
  in IccResource class 170
  in IccTempStore class 217
  in IccTerminal class 225
  in Polymorphic Behavior 56

clear
  in AIDVal 235

clearData
  in IccStartRequestQ class 193

clearInputMessage
  in IccProgram class 157

clearPrefix
  in IccJournal class 139

closed
  in Status 129

cmmCICS
  in ClassMemoryMgmt 65
  in Storage management 58

cmmDefault
  in ClassMemoryMgmt 65
  in Storage management 58

cmmNonCICS
  in ClassMemoryMgmt 65
  in Storage management 58

CODE/370 46

Codes
  in Enumerations 97
  in IccCondition structure 97

col (parameter)
  in send 229
  in send3270 230
  in sendLine 231
  in setCursor 232

Color
  in Enumerations 235
  in IccTerminal class 235
color (parameter)
in operator<< 227
in setColor 231

commArea
in IccControl class 103
commArea (parameter)
in link 158
in setNextCommArea 232

commitOnReturn
in CommitOpt 160
CommitOpt
in Enumerations 160
in IccProgram class 160

commitUOW
in IccTask class 207
Compile and link "Hello World"
in Hello World 10

Compiling, executing, and debugging
Execution Diagnostic Facility 46
Symbolic Debuggers 46
Tracing a Foundation Class Program 46

compiling programs 45

Compiling Programs
in Compiling, executing, and debugging 45

complete
in Kind 149
complete key 28

completeLength
in IccKey class 147
in Public methods 147
completeLength (parameter)
in Constructor 147

condition
in IccEvent class 115
in IccResource class 170
in Manual condition handling (noAction) 50
in Resource classes 17

condition (parameter)
in actionOnCondition 169
in actionOnConditionAsChar 169
in conditionText 63
in setActionOnCondition 172

condition 0 (NORMAL)
in actionsOnConditionsText 170
condition 1 (ERROR)
in actionsOnConditionsText 170
condition 2 (RDATT)
in actionsOnConditionsText 170
condition 3 (WRBRK)
in actionsOnConditionsText 170
condition 4 (ICEEOF)
in actionsOnConditionsText 170
condition 5 (EODS)
in actionsOnConditionsText 170
condition 6 (EOC)
in actionsOnConditionsText 170

Conditions, errors, and exceptions (continued)
Object level 52
Parameter level 53
Severe error handling (abendTask) 52

conditionText
in Functions 63
in Icc structure 63
in IccEvent class 116
in IccResource class 170

ConditionType
in Enumerations 174
in IccResource class 174

confirmation
in SendOpt 191
connectProcess
in IccSession class 184
in Public methods 184

console
in IccControl class 104

const
in Glossary 289

Constructor
in IccAbendData class 67
in IccAbendData constructor (protected) 67
in IccAbsTime class 73
in IccAbsTime constructor 73
in IccAlarmRequestId class 77
in IccAlarmRequestId constructors 77
in IccBase class 79
in IccBase constructor (protected) 79
in IccBuf class 83, 84
in IccBuf constructors 83, 84
in IccClock class 91
in IccClock constructor 91
in IccConsole class 99
in IccConsole constructor (protected) 99
in IccControl class 103
in IccControl constructor (protected) 103
in IccConvId class 107
in IccConvId constructors 107
in IccDataQueue class 109
in IccDataQueue constructors 109
in IccDataQueueId class 113
in IccDataQueueId constructors 113
in IccEvent class 115
in IccEvent constructor 115
in IccException class 117
in IccException constructor 117
in IccFile class 121
in IccFile constructors 121
in IccFileId class 131
in IccFileId constructors 131
in IccFileIterator class 133
in IccFileIterator constructor 133
in IccGroupId class 137
in IccGroupId constructors 137
in IccJournal class 139
in IccJournal constructors 139
in IccJournalId class 143
in IccJournalId constructors 143
in IccJournalTypeId class 145
Constructor (continued)
in IccJournalTypeld constructors 145
in IccKey class 147
in IccKey constructors 147
in IccLockId class 151
in IccLockId constructors 151
in IccMessage class 153
in IccMessage constructor 153
in IccPartnerId class 155
in IccPartnerId constructors 155
in IccProgram class 157
in IccProgram constructors 157
in IccProgramId class 161
in IccProgramId constructors 161
in IccRBA class 163
in IccRBA constructor 163
in IccRecordIndex class 165
in IccRecordIndex constructor (protected) 165
in IccRequestId class 167
in IccRequestld constructors 167
in IccResource class 169
in IccResource constructor (protected) 169
in IccResourceld class 175
in IccResourceld constructors (protected) 175
in IccRRN class 177
in IccRRN constructors 177
in IccSemaphore class 179
in IccSemaphore constructor 179
in IccSession class 183
in IccSession constructor (protected) 183
in IccSession constructors (public) 183
in IccStartRequestQ class 193
in IccStartRequestQ constructor (protected) 193
in IccSysld class 199
in IccSysId class 199
in IccSystem class 201
in IccSystem constructor (protected) 201
in IccTask class 207
in IccTask Constructor (protected) 207
in IccTempStore class 217
in IccTempStore class 217
in IccTempStoreld class 221
in IccTempStoreld constructors 221
in IccTermId class 223
in IccTermId constructors 223
in IccTerminal class 225
in IccTerminal constructor (protected) 225
in IccTerminalData class 237
in IccTerminalData constructor (protected) 237
in IccTime class 243
in IccTime constructor (protected) 243
in IccTimeInterval class 247
in IccTimeInterval constructors 247
in IccTimeOfDay class 249
in IccTimeOfDay constructors 249
in IccTPNameId class 251
in IccTPNameId constructors 251
in IccTransId class 253
in IccTransId constructors 253
in IccUser class 255
in IccUser constructors 255

Constructor (continued)
in IccUserId class 259
in IccUserId constructors 259
converse
in IccSession class 185
convl
in IccSession class 185
convl (parameter)
in Constructor 107
convName (parameter)
in Constructor 107
in operator= 107
copt (parameter)
in setStartOpts 196
createDump
in AbendDumpOpt 214
creating a resource object
in Overview of the foundation classes 19
in Using CICS resources 19
in Singleton classes 20
Creating an object
in C++ Objects 13
creating object 13
current (parameter)
in setPrefix 140
cursor
in Finding out information about a terminal 42
in IccTerminal class 225
customClassNum
in IccBase class 79
in Public methods 79
cut
in IccBuf class 85
in IccBuf constructors 24
CVDA
in Enumeration 261
in IccValue structure 261
cyan
in Color 235

D
data
in Accessing start data 34
in Finding out information about a terminal 42
in IccStartRequestQ class 194
in IccTerminal class 225
in IccTerminalData class 237
data (parameter)
in enterTrace 208, 209
in put 188
data area extensibility 23
Data area extensibility
in Buffer objects 23
in IccBuf class 23
data area ownership 23
Data area ownership
in Buffer objects 23
in IccBuf class 23
dataArea
  in IccBuf class 85
dataArea (parameter)
  in append 84
  in assign 84, 147
  in Constructor 83
  in insert 86
  in overlay 89
  in replace 89
dataAreaLength
  in IccBuf class 85
  in Public methods 85
dataAreaOwner
  in Data area ownership 23
  in IccBuf class 85
DataAreaOwner
  in Enumerations 90
  in IccBuf class 90
dataAreaType
  in Data area extensibility 23
  in IccBuf class 85
DataAreaType
  in Enumerations 90
  in IccBuf class 90
dataItems
  in Example of polymorphic behavior 56
dataLength
  in IccBuf class 85
dataqueue
  in FacilityType 215
dataQueueTrigger
  in StartType 215
date
  in IccAbsTime class 73
  in IccClock class 91
date services 43
dateFormat
  in IccSystem class 202
DateFormat
  in Enumerations 94
  in IccClock class 94
dateSeparator (parameter)
  in date 73, 92
  in Example of time and date services 44
dayOfMonth
  in Example of time and date services 44
  in IccAbsTime class 73
  in IccClock class 92
dayOfWeek
  in Example of time and date services 44
  in IccAbsTime class 73
  in IccClock class 92
dayOfWeek
  in Enumerations 94
  in IccClock class 94
daysSince1900
  in Example of time and date services 44
  in IccAbsTime class 74
  in IccClock class 92
daysUntilPasswordExpires
  in IccUser class 256
dComplete
  in DumpOpts 214
dDCT
  in DumpOpts 215
dDefault
  in DumpOpts 214
debuggers 46
debugging programs 45
Debugging Programs
  Activating the trace output 46
  Enabling EDF 46
  Execution Diagnostic Facility 46
  in Compiling, executing, and debugging 45
  Symbolic Debuggers 46
  Tracing a Foundation Class Program 46
defaultColor
  in Color 235
defaultHeight
  in IccTerminalData class 237
  in Public methods 237
defaultHighlight
  in Highlight 235
defaultWidth
  in IccTerminalData class 238
  in Public methods 238
delay
  in IccTask class 208
  in Support Classes 19
deletable
  in Access 129
delete
  in Deleting an object 14
  in Storage management 58
delete operator 13
deleteLockedRecord
  in Deleting locked records 30
  in IccFile class 122
deleteRecord
  in Deleting normal records 29
  in IccFile class 122
deleteRecord method 29
Deleting an object
  in C++ Objects 14
deleting items 40
Deleting items
  in Temporary storage 40
  in Using CICS Services 40
Deleting locked records
  in Deleting records 30
  in File control 30
Deleting normal records
  in Deleting records 29
  in File control 29
deleting queues 38
Deleting queues
  in Transient Data 38
  in Using CICS Services 38
deleting records 29
Deleting records
  Deleting locked records 30
  Deleting normal records 29
Deleting records (continued)
  in File control 29
  in Using CICS Services 29
dFCT
  in DumpOpts 215
DFHCURDI 7
DFHCURDS 6, 7
disabled
  in Status 130
doSomething
  in Using an object 14
dPCT
  in DumpOpts 215
DPL
  in StartType 215
dPPT
  in DumpOpts 215
dProgram
  in DumpOpts 215
dSIT
  in DumpOpts 215
dStorage
  in DumpOpts 214
dTables
  in DumpOpts 215
dTask
  in DumpOpts 214
dTCT
  in DumpOpts 215
dTerminal
  in DumpOpts 215
dTRT
  in DumpOpts 215
dump
  in IccTask class 208
dumpCode (parameter)
  in dump 208
DumpOpts
  in Enumerations 214
  in IccTask class 214
dynamic creation 13
dynamic deletion 13
dynamic link library 6
Dynamic link library
  in Installed contents 6
  Location 6

E
ECBList (parameter)
  in waitExternal 213
EDF (Execution Diagnostic Facility) 46
EDF (parameter)
  in initializeEnvironment 64
empty
  in Deleting items 40
  in Deleting queues 38
  in IccDataQueue class 109
  in IccTempStore class 217
  in Temporary storage 39
  in Transient Data 37

enabled
  in Status 129
enableStatus
  in IccFile class 122
Enabling EDF
  in Debugging Programs 46
  in Execution Diagnostic Facility 46
deBrowse
  in IccSystem class 202
dInsert
  in Writing records 28
dInsert(VSAM only)
  in IccFile class 123
  in Public methods 123
endl
  in Example of terminal control 43
ENTER
  in AIDVal 235
enterTrace
  in IccTask class 208
entryPoint
  in IccProgram class 158
Enumeration
  CVDA 261
  in IccValue structure 261
Enumerations
  AbendDumpOpt 214
  AbendHandlerOpt 214
  Access 129
  ActionOnCondition 173
  AIDVal 235
  AllocateOpt 191
  Bool 65
  BoolSet 65
  Case 235
  CheckOpt 198
  ClassMemoryMgmt 65
  ClassType 80
  Codes 97
  Color 235
  CommitOpt 160
  ConditionType 174
  DataAreaOwner 90
  DataAreaType 90
  DateFormat 94
  DayOfWeek 94
  DumpOpts 214
  FacilityType 215
  FamilySubset 66
  GetOpt 66
  HandleEventReturnOpt 174
  Highlight 235
  in Icc structure 65
  in IccBase class 80
  in IccBuf class 90
  in IccClock class 94
  in IccCondition structure 97
  in IccConsole class 102
  in IccException class 119
  in IccFile class 129
  in IccJournal class 142
Enumerations (continued)
in IccKey class 149
in IccProgram class 160
in IccRecordIndex class 166
in IccSemaphore class 181
in IccSession class 191
in IccStartRequestQ class 198
in IccSystem class 205
in IccTask class 214
in IccTempStore class 220
in IccTerminal class 235
in IccTime class 244
Kind 149
LifeTime 181
LoadOpt 160
Location 220
LockType 181
MonthOfYear 94
NameOpt 81
NextTransIdOpt 235
NoSpaceOpt 220
Options 142
Platforms 66
ProtectOpt 198
Range 97
ReadMode 129
ResourceType 205
RetrieveOpt 198
SearchCriterion 129
SendOpt 191
SeverityOpt 102
StartType 215
StateOpt 191
Status 129
StorageOpts 215
SyncLevel 191
TraceOpt 215
Type 119, 166, 244
UpdateMode 95
WaitPostType 216
WaitPurgeability 216
equalToKey
in SearchCriterion 129
erase
in Example of terminal control 43
in Hello World 9
in IccTerminal class 226
in Sending data to a terminal 41
errorCode
in IccSession class 185
ESDS
in File control 27
ESDS file 27
ESMReason
in IccUser class 256
ESMResponse
in IccUser class 256
event (parameter)
in handleEvent 171
Example of file control
in File control 30
in Using CICS Services 30
Example of managing transient data
in Transient Data 38
in Using CICS Services 38
Example of polymorphic behavior
in Miscellaneous 56
in Polymorphic Behavior 56
Example of starting transactions
in Starting transactions asynchronously 34
in Using CICS Services 34
Example of Temporary Storage
in Temporary storage 40
in Using CICS Services 40
Example of terminal control
in Terminal control 42
in Using CICS Services 42
Example of time and date services
in Time and date services 43
in Using CICS Services 43
exception
in TraceOpt 215
exception (parameter)
in catchException 63
Exception handling (throwException)
in CICS conditions 51
in Conditions, errors, and exceptions 51
exceptionNum (parameter)
in Constructor 117
exceptions 47
exceptionType (parameter)
in Constructor 117
Executing Programs
in Compiling, executing, and debugging 45
Execution Diagnostic Facility
Enabling EDF 46
in Compiling, executing, and debugging 46
in Debugging Programs 46
Execution Diagnostic Facility (EDF) 46
Expected Output from "Hello World"
in Hello World 10
in Running "Hello World" on your CICS server 10
testable
in DataAreaType 90
external
in DataAreaOwner 90
extractProcess
in IccOwner class 185
extractState
in StateOpt 191

F

facilitype
in IccTask class 209

FACILITYPE
in Enumerations 215

fam (parameter)
in initializeEnvironment 64
familyConformanceError
  in C++ Exceptions and the Foundation Classes 49
  in Type 120
FamilySubset
  in Enumerations 66
  in Icc structure 66
FEPIRequest
  in StartType 215
file (parameter)
  in Constructor 133
  in Example of file control 31
file control
  browsing records 30
  deleting records 29
  example 30
  rewriting records 29
  updating records 29
File control
  Browsing records 30
  Deleting locked records 30
  Deleting normal records 29
  Deleting records 29
  Example of file control 30
  in Using CICS Services 27
  Reading ESDS records 28
  Reading KSDDS records 28
  Reading records 27
  Reading RRDS records 28
  Updating records 29
  Writing ESDS records 29
  Writing KSDDS records 29
  Writing records 28
  Writing RRDS records 29
fileName (parameter)
  in Constructor 121, 131
  in getFile 202
  in operator= 131
Finding out information about a terminal
  in Terminal control 42
  in Using CICS Services 42
First Screen
  in ICC$PRG1 (IPR1) 284
  in Output from sample programs 284
fixed
  in DataAreaType 90
flush
  in Example of terminal control 43
  in IccSession class 186
for
  in Example of file control 31
Form
  in Polymorphic Behavior 55
format (parameter)
  in append 84
  in assign 85
  in date 73, 92
  in Example of time and date services 44
  in send 229
  in send3270 230
  in sendLine 231
Foundation Class Abend codes
  in Conditions, errors, and exceptions 47
free
  in IccSession class 186
freeKeyboard
  in IccTerminal class 226
  in Sending data to a terminal 41
freeStorage
  in IccSystem class 202
  in IccTask class 209
fsAllowPlatformVariance
  in FamilySubset 66
  in Platform differences 52
fsDefault
  in FamilySubset 66
fsEnforce
  in FamilySubset 66
  in Platform differences 52
fullAccess
  in Access 129
Functions
  boolText 63
  catchException 63
  conditionText 63
  in Icc structure 63
  initializeEnvironment 63
  isClassMemoryMgmtOn 64
  isEDFOn 64
  isFamilySubsetEnforcementOn 64
  returnToCICS 64
  setEDF 64
  unknownException 65
G
generic
  in Kind 149
  generic key 28
get
  in Example of polymorphic behavior 57
  in IccDataQueue class 109
  in IccResource class 170
  in IccSession class 186
  in IccTempStore class 218
  in IccTerminal class 226
  in Polymorphic Behavior 56
getFile
  in IccSystem class 202
getNextFile
  in IccSystem class 203
GetOpt
  in Enumerations 66
  in Icc structure 66
getStorage
  in IccSystem class 203
  in IccTask class 209
gid (parameter)
  in Constructor 255
graphicCharCodeSet
  in IccTerminalData class 238
graphicCharSetId
  in IccTerminalData class 238
green
  in Color 235
groupId
  in IccUser class 256
groupName (parameter)
  in Constructor 137, 255
  in operator= 137
gteqToKey
  in SearchCriterion 129

H
  H
  in actionOnConditionAsChar 169
handleEvent
  in Automatic condition handling (callHandleEvent) 50, 51
  in IccResource class 171
HandleEventReturnOpt
  in Enumerations 174
  in IccResource class 174
handPost
  in WaitPostType 216
Header files
  in Installed contents 5
  Location 6
height
  in IccTerminal class 226
Hello World
  commentary 9
  Compile and link 10
  Expected Output from "Hello World" 10
  running 10
Highlight
  in Enumerations 235
  in IccTerminal class 235
highlight (parameter)
  in operator<< 227
  in setHighlight 232
hold
  in LoadOpt 160
hours
  in IccAbsTime class 74
  in IccTime class 243
hours (parameter)
  in Constructor 243, 247, 249
  in set 247, 249

I
  Icc
    in Foundation Classes—reference 62
    in Method level 53
    in Overview of the foundation classes 15
    Icc::initializeEnvironment
      in Storage management 58
    ICC$BUF 6
    ICC$BUF (IBUF)
      in Output from sample programs 281
    ICC$CLK 6
    ICC$CLK (ICLK)
      in Output from sample programs 281
    ICC$DAT (IDAT)
      in Output from sample programs 281
    ICC$EXC1 (IEX1)
      in Output from sample programs 282
    ICC$EXC2 (IEX2)
      in Output from sample programs 282
    ICC$EXC3 (IEX3)
      in Output from sample programs 282
    ICC$FIL (IFIL)
      in Output from sample programs 283
    ICC$HEL 6
    ICC$HEL (IHEL)
      in Output from sample programs 283
    ICC$JRN (IJRN)
      in Output from sample programs 283
    ICC$PRG1 (IPR1)
      First Screen 284
      in Output from sample programs 284
      Second Screen 284
    ICC$RES1 (IRE1)
      in Output from sample programs 284
    ICC$RES2 (IRE2)
      in Output from sample programs 285
    ICC$SEM (ISEM)
      in Output from sample programs 285
    ICC$SES1 6
    ICC$SES1 (ISE1)
      in Output from sample programs 285
    ICC$SES2 6
      in Output from sample programs 286
    ICC$SRQ1 (ISR1)
      in Output from sample programs 286
    ICC$SRQ2 (ISR2)
      in Output from sample programs 286
    Icc structure
      Bool 65
      BoolSet 65
      boolText 63
      catchException 63
      ClassMemoryMgmt 65
      conditionText 63
      FamilySubset 66
      GetOpt 66
      initializeEnvironment 63
      isClassMemoryMgmtOn 64
      isEDFOn 64
      isFamilySubsetEnforcementOn 64
      Platforms 66
      returnToCICS 64
      setEDF 64
      unknownException 65
    ICC$SYS (ISYS)
      in Output from sample programs 287
    ICC$TMP (ITMP)
      in Output from sample programs 287
    ICC$TRM (ITRM)
      in Output from sample programs 287
ICC$TSK (ITSK)
in Output from sample programs 288
IccAbendData
 in Singleton classes 20
IccAbendData class
  abendCode 67
  ASRAInterrupt 67
  ASRAKeyType 68
  ASRAPSW 68
  ASRAResisters 68
  ASRASpaceType 69
  ASRAStorageType 69
  Constructor 67
  instance 70
  isDumpAvailable 70
  originalAbendCode 70
  programName 70
IccAbendData constructor (protected)
  Constructor 67
  in IccAbendData class 67
IccAbsTime
  in Base classes 16
  in delay 208
  in IccTime class 243
  in Support Classes 19
  in Time and date services 43
IccAbsTime,
  in Support Classes 19
IccAbsTime class
  Constructor 73
  date 73
  dayOfMonth 73
  dayOfWeek 74
  daysSince1900 74
  hours 74
  milliSeconds 74
  minutes 74
  monthOfYear 74
  operator= 74
  packedDecimal 75
  seconds 75
  time 75
  timeInHours 75
  timeInMinutes 75
  timeInSeconds 75
  year 75
IccAbsTime constructor
  Constructor 73
  in IccAbsTime class 73
IccAlarmRequestId
  in IccAlarmRequestId class 77
IccAlarmRequestId class
  Constructor 77
  isExpired 77
  operator= 78
  setTimerECA 78
  timerECA 78
IccAlarmRequestId constructors
  Constructor 77
  in IccAlarmRequestId class 77
IccBase
  in Base classes 15
  in Foundation Classes—reference 62
  in IccAbendData class 67
  in IccAbsTime class 73
  in IccAlarmRequestId class 77
  in IccBase class 79
  in IccBuf class 83
  in IccClock class 91
  in IccConsole class 99
  in IccControl class 103
  in IccConvId class 107
  in IccDataQueue class 109
  in IccDataQueueId class 113
  in IccEvent class 115
  in IccException class 117
  in IccFile class 121
  in IccFieldId class 131
  in IccFileIterator class 133
  in IccGroupId class 137
  in IccJournal class 139
  in IccJournalId class 143
  in IccJournalTypelId class 145
  in IccKey class 147
  in IccLockId class 151
  in IccMessage class 153
  in IccPartnerId class 155
  in IccProgram class 157
  in IccProgramId class 161
  in IccRBA class 163
  in IccRecordIndex class 165
  in IccRequestlId class 167
  in IccResource class 169
  in IccResourceId class 175
  in IccRRN class 177
  in IccSemaphore class 179
  in IccSession class 183
  in IccStartRequestQ class 193
  in IccSysId class 199
  in IccSystem class 201
  in IccTask class 207
  in IccTempStore class 217
  in IccTempStoreId class 221
  in IccTermId class 223
  in IccTerminal class 225
  in IccTerminalData class 237
  in IccTime class 243
  in IccTimeInterval class 247
  in IccTimeOfDay class 249
  in IccTPNameId class 251
  in IccTransId class 253
  in IccUser class 255
  in IccUserId class 259
  in Resource classes 17
  in Resource identification classes 16
  in Storage management 58
  in Support Classes 18
IccBase class
  className 79
  classType 79
  ClassType 80
IccBase class (continued)
Constructor 79
customClassNum 79
NameOpt 81
operator delete 80
operator new 80
overview 15
setClassName 80
setCustomClassNum 80

IccBase constructor (protected)
Constructor 79
in IccBase class 79

IccBuf
in Buffer objects 23
in C++ Exceptions and the Foundation Classes 49
in Data area extensibility 23
in Data area ownership 23
in Example of file control 31
in Example of managing transient data 38
in Example of polymorphic behavior 56
in Example of starting transactions 35, 36, 37
in Example of Temporary Storage 40, 41
in Example of terminal control 42
in IccBuf class 23, 83
in IccBuf constructors 23, 24
in IccBuf methods 24
in Reading data 37
in Reading items 39
in Scope of data in IccBuf reference returned from ‘read’ methods 59
in Support Classes 19
in Working with IccResource subclasses 25

IccBuf class
append 84
assign 84, 85
Constructor 83, 84
constructors 23
cut 85
data area extensibility 23
Data area extensibility 23
data area ownership 23
Data area ownership 23
dataArea 85
dataAreaLength 85
dataAreaOwner 85
DataAreaOwner 90
dataAreaType 85
DataAreaType 90
dataLength 85
IccBuf constructors 23
IccBuf methods 24
in Buffer objects 23
insert 86
isFMHContained 86
methods 24
operator= 86
operator!= 87
operator++ 87
operator-- 87
operator const char* 86
operator<< 87, 88

IccBuf class (continued)
overlay 89
replace 89
setDataLength 89
setFMHContained 89
Working with IccResource subclasses 24

IccBuf constructors 23
Constructor 83, 84
in Buffer objects 23
in IccBuf class 23, 83

IccBuf methods 24
in Buffer objects 24
in IccBuf class 24

IccBuf reference 59

IccClock
in Example of time and date services 43, 44
in IccAlarmRequestId class 77
in IccClock class 91
in Time and date services 43

IccClock class
absTime 91
cancelAlarm 91
Constructor 91
date 91
dateFormat 94
dayOfMonth 92
dayOfWeek 92
DayOfWeek 94
daysSince1900 92
milliSeconds 92
monthOfYear 92
MonthOfYear 94
setAlarm 93
time 93
update 93
UpdateMode 95
year 93

IccClock constructor
Constructor 91
in IccClock class 91

IccCondition
in C++ Exceptions and the Foundation Classes 49

IccCondition structure
Codes 97
Range 97

IccConsole
in Buffer objects 23
in Object level 52, 53
in Singleton classes 20

IccConsole class
Constructor 99
instance 99
overview 20
put 99
replyTimeout 99
resetRouteCodes 100
setAllRouteCodes 100
setReplyTimeout 100
setRouteCodes 100
SeverityOpt 102
write 100
IccConsole class (continued)
        writeAndGetReply 101
IccConsole constructor (protected)
                Constructor 99
                in IccConsole class 99
IccControl
        in Base classes 15
        in Example of starting transactions 35, 36
        in Hello World 9
        in IccControl class 103
        in IccProgram class 157
        in main function 265, 266
        in Mapping EXEC CICS calls to Foundation Class methods 275
        in Method level 53
        in Singleton classes 20
        in Support Classes 19
IccControl::run
        in Mapping EXEC CICS calls to Foundation Class methods 275
IccControl class
        callingProgramId 103
        cancelAbendHandler 103
        commArea 103
        console 104
        Constructor 103
        initData 104
        instance 104
        isCreated 104
        overview 15, 20
        programId 104
        resetAbendHandler 104
        returnProgramId 105
        run 105
        session 105
        setAbendHandler 105
        startRequestQ 105
        system 105
        task 106
        terminal 106
IccControl constructor (protected)
                Constructor 103
                in IccControl class 103
IccConvlId
        in IccConvlId class 107
IccConvlId class
        Constructor 107
        operator= 107
IccConvlId constructors
        Constructor 107
        in IccConvlId class 107
IccDataQueue class
        clear 109
        Constructor 109
        empty 109
        get 109
        put 110
        readItem 110
        writeItem 110
IccDataQueue constructors
        Constructor 109
        in IccDataQueue class 109
IccDataQueueId
        in Example of managing transient data 38
        in IccDataQueueId class 113
        in Transient Data 37
IccDataQueueId class
        Constructor 113
        operator= 113
IccDataQueueId constructors
        Constructor 113
        in IccDataQueueId class 113
IccEvent
        in IccEvent class 115
        in Support Classes 19
IccEvent class
        className 115
        classType 115
        condition 115
        conditionText 116
        Constructor 115
        methodName 116
        summary 116
IccEvent constructor
        Constructor 115
        in IccEvent class 115
IccException
        in C++ Exceptions and the Foundation Classes 48, 49
        in IccException class 117
        in IccMessage class 153
        in main function 266
        in Method level 53
        in Object level 53
        in Parameter level 53, 54
        in Support Classes 19
IccException class
        CICSCondition type 49
        className 118
        classType 118
        Constructor 117
        familyConformanceError type 49
        internalError type 49
        invalidArgument type 48
        invalidMethodCall type 49
        message 118
        methodName 118
        number 118
        objectCreationError type 48
        summary 118
        type 118
        Type 119
IccException class (continued)
  typeText 119
IccException constructor
  Constructor 117
  in IccException class 117
ICCFCC 6
ICCFCL 6
ICCFCDLL 6
ICCFCIMP 6
ICCFCL 6
IccFile
  in Browsing records 30
  in Buffer objects 23
  in C++ Exceptions and the Foundation Classes 49
  in Deleting locked records 30
  in Deleting normal records 29
  in Example of file control 30
  in File control 27
  in IccFile class 121
  in IccFileIterator class 133
  in Reading ESDS records 28
  in Reading KSDS records 28
  in Reading records 27, 28
  in Reading RRDS records 28
  in Resource identification classes 16
  in Singleton classes 20
  in Updating records 29
  in Writing ESDS records 29
  in Writing KSDS records 29
  in Writing records 28
  in Writing RRDS records 29
IccFile::readRecord
  in Scope of data in IccBuf reference returned from
    'read' methods 59
IccFile class
  access 121
  Access 129
  accessMethod 122
  beginInsert(VSAM only) 122
  Constructor 121
  deleteLockedRecord 30, 122
  deleteRecord 122
  deleteRecord method 29
  enableStatus 122
  endInsert(VSAM only) 123
  isAddable 123
  isBrowsable 123
  isDeleteable 123
  isReadable 124
  isReadable method 28
  isRecoverable 124
  isUpdatable 124
  keyLength 124
  keyLength method 28
  keyPosition 124
  keyPosition method 28
  openStatus 125
  ReadMode 129
  readRecord 125
  readRecord method 28
IccFile class (continued)
  recordFormat 125
  recordFormat method 28
  recordIndex 126
  recordIndex method 28
  recordLength 126
  recordLength method 28
  registerRecordIndex 28, 126
  registerRecordIndex method 28
  rewriteRecord 126
  rewriteRecord method 29
  SearchCriterion 129
  setAccess 127
  setEmptyOnOpen 127
  setStatus 127
  Status 129
  type 127
  unlockRecord 128
  writeRecord 128
  writeRecord method 28
IccFile constructors
  Constructor 121
  in IccFile class 121
IccFileId
  in Base classes 15
  in File control 27
  in IccFileId class 131
  in Resource identification classes 16
IccFileId class
  Constructor 131
  operator= 131
  overview 15, 27
  reading records 27
IccFileId constructors
  Constructor 131
  in IccFileId class 131
IccFileIterator
  in Browsing records 30
  in Buffer objects 23
  in Example of file control 30, 31
  in File control 27
  in IccFileIterator class 133
IccFileIterator class
  Constructor 133
  overview 27
  readNextRecord 133
  readNextRecord method 30
  readPreviousRecord 30, 134
  reset 134
IccFileIterator constructor
  Constructor 133
  in IccFileIterator class 133
IccGroupId
  in IccGroupId class 137
IccGroupId class
  Constructor 137
  operator= 137
IccGroupId constructors
  Constructor 137
  in IccGroupId class 137
IccJournal
  in Buffer objects 23
  in IccJournal class 139
  in Object level 52, 53
IccJournal class
  clearPrefix 139
  Constructor 139
  journalTypeId 140
  Options 142
  put 140
  registerPrefix 140
  setJournalTypeId 140
  setPrefix 140
  wait 140
  writeRecord 141
IccJournal constructors
  Constructor 139
  in IccJournal class 139
IccJournalId
  in IccJournalId class 143
IccJournalId class
  Constructor 143
  number 143
  operator= 143
IccJournalId constructors
  Constructor 143
  in IccJournalId class 143
IccJournalTypeId
  in Foundation Classes—reference 62
  in IccJournalTypeId class 145
IccJournalTypeId class
  Constructor 145
  operator= 145
IccJournalTypeId constructors
  Constructor 145
  in IccJournalTypeId class 145
IccKey
  in Browsing records 30
  in Deleting normal records 29
  in File control 27
  in IccKey class 147
  in IccRecordIndex class 165
  in Reading KSDS records 28
  in Reading records 27
  in Writing KSDS records 29
  in Writing records 28
IccKey class 28
  assign 147
  completeLength 147
  Constructor 147
  kind 147
  Kind 149
  operator= 148
  operator!= 148
  operator== 148
  reading records 27
  setKind 148
  value 148
IccKey constructors
  Constructor 147
  in IccKey class 147
IccLockId
  in IccLockId class 151
IccLockId class
  Constructor 151
  operator= 151
IccLockId constructors
  Constructor 151
  in IccLockId class 151
IccMessage
  in IccMessage class 153
  in Support Classes 19
IccMessage class
  className 153
  Constructor 153
  methodName 153
  number 153
  summary 153
  text 154
IccMessage constructor
  Constructor 153
  in IccMessage class 153
IccPartnerId
  in IccPartnerId class 155
IccPartnerId class
  Constructor 155
  operator= 155
IccPartnerId constructors
  Constructor 155
  in IccPartnerId class 155
IccProgram
  in Buffer objects 23
  in IccProgram class 157
  in Program control 32, 33
  in Resource classes 17
IccProgram class
  address 157
  clearInputMessage 157
  CommitOpt 160
  Constructor 157
  entryPoint 158
  length 158
  link 158
  load 158
  LoadOpt 160
  program control 32
  setInputMessage 159
  unload 159
IccProgram constructors
  Constructor 157
  in IccProgram class 157
IccProgramId
  in IccProgramId class 161
  in Resource identification classes 16
IccProgramId class
  Constructor 161
  operator= 161
IccProgramId constructors
  Constructor 161
  in IccProgramId class 161
IccRBA
  in Browsing records 30
IccRBA (continued)
in File control 27
in IccRBA class 163
in IccRecordIndex class 165
in Reading ESDS records 28
in Reading records 27
in Writing ESDS records 29
in Writing records 28
in Writing RRDS records 29

IccRBA class
Constructor 163
number 164
operator= 163
operator!= 163
operator== 163
reading records 27

IccRBA constructor
Constructor 163
in IccRBA class 163

IccRecordIndex
in C++ Exceptions and the Foundation Classes 49
in IccRecordIndex class 165

IccRecordIndex class
Constructor 165
length 165
type 165
Type 166

IccRecordIndex constructor (protected)
Constructor 165
in IccRecordIndex class 165

IccRequest
in Example of starting transactions 35, 36
in IccRequest class 167
in Parameter passing conventions 59

IccRequest class
Constructor 167
operator= 167

IccRequest constructor (protected)
Constructor 167
in IccRequest class 167

IccResourceId
in Base classes 15
in C++ Exceptions and the Foundation Classes 49
in Resource identification classes 16

IccResourceId class
Constructor 175
name 175
nameLength 175
operator= 176
overview 15, 16

IccResourceId constructors (protected)
Constructor 175
in IccResourceId class 175

IccRRN
in Browsing records 30
in Deleting normal records 29
in File control 27
in IccRecordIndex class 165
in IccRRN class 177
in Reading records 27
in Reading RRDS records 28
in Writing records 28

IccRRN class
Constructor 177
number 178
operator= 177
operator!= 177
operator== 177
reading records 27

IccRRN constructors (protected)
Constructor 177
in IccRRN class 177

IccSemaphore class
Constructor 179
lifeTime 179
LifeTime 181
lock 180
LockType 181
tryLock 180
type 180
unlock 180

IccSemaphore constructor
Constructor 179
in IccSemaphore class 179
IccSession
  in Buffer objects  23
IccSession class
  allocate 184
  AllocateOpt 191
  connectProcess 184
  Constructor 183
  converse 185
  convId 185
  errorCode 185
  extractProcess 185
  flush 186
  free 186
  get 186
  isErrorSet 186
  isNoDataSet 186
  isSignalSet 186
  issueAbend 186
  issueConfirmation 187
  issueError 187
  issuePrepare 187
  issueSignal 187
  PIPList 187
  process 187
  put 188
  receive 188
  send 188
  sendInvite 188, 189
  sendLast 189
  SendOpt 191
  state 189
  StateOpt 191
  stateText 190
  syncLevel 190
  SyncLevel 191
IccSession constructor (protected)
  Constructor 183
  in IccSession class 183
IccSession constructors (public)
  Constructor 183
  in IccSession class 183
IccStartRequestQ
  in Accessing start data 34
  in Buffer objects 23
  in Example of starting transactions 35, 36
  in IccRequestId class 167
  in IccStartRequestQ class 193
  in Mapping EXEC CICS calls to Foundation Class methods 275
  in Parameter passing conventions 58
  in Singleton classes 20
  in Starting transactions asynchronously 34
IccStartRequestQ class
  cancel 193
  CheckOpt 198
  clearData 193
  Constructor 193
  data 194
  instance 194
  overview 20
  ProtectOpt 198
IccStartRequestQ class (continued)
  queueName 194
  registerData 194
  reset 194
  retrieveData 194
  RetrieveData 194
  returnTermId 195
  returnTransId 195
  setData 195
  setQueueName 195
  setReturnTermId 195
  setReturnTransId 196
  setStartOpts 196
  start 196
IccStartRequestQ constructor (protected)
  Constructor 193
  in IccStartRequestQ class 193
IccSysId
  in IccSysId class 199
  in Program control 33
IccSysId class
  Constructor 199
  operator= 199
IccSysId constructors
  Constructor 199
  in IccSysId class 199
IccSystem
  in Singleton classes 20
IccSystem class
  applName 201
  beginBrowse 201
  Constructor 201
  dateFormat 202
  endBrowse 202
  freeStorage 202
  getFile 202
  getNextFile 203
  getStorage 203
  instance 203
  operatingSystem 203
  operatingSystemLevel 203
  overview 20
  release 204
  releaseText 204
  ResourceType 205
  sysId 204
  workArea 204
IccSystem constructor (protected)
  Constructor 201
  in IccSystem class 201
IccTask
  in C++ Exceptions and the Foundation Classes 48
  in Example of starting transactions 36
  in IccAlarmRequestId class 77
  in IccTask class 207
  in Parameter level 53
  in Singleton classes 20
  in Support Classes 19
IccTask::commitUOW
  in Scope of data in IccBuf reference returned from
  'read' methods 59
IccTask class
  abend 207
  abendData 207
  AbendDumpOpt 214
  AbendHandlerOpt 214
  commitUOW 207
  Constructor 207
  delay 208
  dump 208
  DumpOpts 214
  enterTrace 208
  facilityType 209
  FacilityType 215
  freeStorage 209
  getStorage 209
  instance 210
  isCommandSecurityOn 210
  isCommitSupported 210
  isResourceSecurityOn 210
  isRestarted 210
  isStartDataAvailable 210
  number 211
  overview 20
  principalSysId 211
  priority 211
  rollbackUOW 211
  setDumpOpts 211
  setPriority 211
  setWaitText 212
  startType 212
  StartType 215
  StorageOpts 215
  suspend 212
  TraceOpt 215
  transId 212
  triggerDataQueueId 212
  userid 212
  waitExternal 213
  waitOnAlarm 213
  WaitPostType 216
  WaitPurgeability 216
  workArea 213
IccTask Constructor (protected)
  Constructor 207
  in IccTask class 207
IccTempStore
  in Working with IccResource subclasses 24
IccTempStore (continued)
  in Working with IccResource subclasses 25
  in Writing items 39
IccTempStore::readItem
  in Scope of data in IccBuf reference returned from 'read' methods 59
IccTempStore::readNextItem
  in Scope of data in IccBuf reference returned from 'read' methods 59
IccTempStore class
  clear 217
  Constructor 217
  empty 217
  get 218
  Location 220
  NoSpaceOpt 220
  numberOfItems 218
  put 218
  readItem 218
  readNextItem 218
  rewriteItem 219
  writeItem 219
IccTempStore constructors
  Constructor 217
  in IccTempStore class 217
IccTempStoreId
  in Base classes 15
  in Example of Temporary Storage 40
  in IccTempStoreId class 221
  in Temporary storage 39
IccTempStoreId class
  Constructor 221
  operator= 221
IccTempStoreId constructors
  Constructor 221
  in IccTempStoreId class 221
IccTermId
  in Base classes 15
  in C++ Exceptions and the Foundation Classes 49
  in Example of starting transactions 35
  in Example of terminal control 42
  in IccTermId class 223
  in Terminal control 41
IccTermId class
  Constructor 223
  operator= 223
  overview 15
IccTermId constructors
  Constructor 223
  in IccTermId class 223
IccTerminal
  in Buffer objects 23
  in Example of terminal control 42
  in Finding out information about a terminal 42
  in IccTerminalData class 237
  in Receiving data from a terminal 42
  in Resource classes 17, 18
  in Sending data to a terminal 41
  in Singleton classes 20
  in Terminal control 41
IccTerminal::receive
  in Scope of data in IccBuf reference returned from 'read' methods 59

IccTerminal class
  AID 225
  AIDVal 235
  Case 235
  clear 225
  Color 235
  Constructor 225
  cursor 225
  data 225
  erase 226
  freeKeyboard 226
  get 226
  height 226
  Highlight 235
  inputCursor 226
  instance 226
  line 226
  netName 227
  NextTransIdOpt 235
  operator<< 227, 228
  put 228
  receive 228
  receive3270Data 229
  registerInputMessage 159
  send 229
  send3270 230
  sendLine 230, 231
  setColor 231
  setCursor 231, 232
  setHighlight 232
  setLine 232
  setNewLine 232
  setNextCommArea 232
  setNextInputMessage 233
  setNextTransId 233
  signoff 233
  signon 233
  waitForAID 234
  width 234
  workArea 234

IccTerminal constructor (protected)
  Constructor 225
  in IccTerminal class 225

IccTerminalData
  in Example of terminal control 42
  in Finding out information about a terminal 42
  in IccTerminalData class 237
  in Terminal control 41

IccTerminalData class (continued)
  isBTrans 238
  isColor 239
  isEWA 239
  isExtended3270 239
  isFieldOutline 239
  isGoodMorning 239
  isHighlight 240
  isKatakana 240
  isMSRControl 240
  isPS 240
  isSOSI 240
  isTextKeyboard 240
  isTextPrint 241
  isValidation 241

IccTerminalData constructor (protected)
  Constructor 237
  in IccTerminalData class 237

IccTime
  in Base classes 16
  in IccTime class 243
  in Parameter passing conventions 59
  in Support Classes 19

IccTime class
  Constructor 243
  hours 243
  minutes 243
  overview 16
  seconds 243
  timeInHours 243
  timeInMinutes 244
  timeInSeconds 244
  type 244
  Type 244

IccTime constructor (protected)
  Constructor 243
  in IccTime class 243

IccTimeInterval
  in Base classes 16
  in delay 208
  in Example of starting transactions 35, 36
  in IccTime class 243
  in Support Classes 19

IccTimeInterval class
  Constructor 247
  operator= 247
  set 247

IccTimeInterval constructors
  Constructor 247
  in IccTimeInterval class 247

IccTimeOfDay
  in Base classes 16
  in delay 208
  in IccTime class 243
  in Support Classes 19

IccTimeOfDay class
  Constructor 249
  operator= 249
  set 249

IccTimeOfDay constructors
  Constructor 249
null
Inherited protected methods (continued)
in IccRRN class 178
in IccSemaphore class 181
in IccSession class 191
in IccStartRequestQ class 197
in IccSysId class 200
in IccSystem class 205
in IccTask class 214
in IccTempStore class 220
in IccTempStoreId class 222
in IccTermId class 224
in IccTerminal class 235
in IccTerminalData class 241
in IccTime class 244
in IccTimeInterval class 248
in IccTimeOfDay class 250
in IccTPNameId class 252
in IccUser class 253
in IccUserId class 257
in IccUserld class 259
initByte (parameter)
in getStorage 203, 209
initData
in IccControl class 104
in Public methods 104
initializeEnvironment
in Functions 63
in Icc structure 63
in Method level 53
in Storage management 58
initRBA (parameter)
in Constructor 163
initRRN (parameter)
in Constructor 177
initValue (parameter)
in Constructor 147
inputCursor
in IccTerminal class 226
insert
in Example of Temporary Storage 41
in IccBuf class 86
in IccBuf constructors 24
Installed contents
Location 6
instance
in IccAbendData class 70
in IccConsole class 99
in IccControl class 104
in IccStartRequestQ class 194
in IccSystem class 203
in IccTask class 210
in IccTerminal class 226
in Singleton classes 20
internal
in DataAreaOwner 90
internalError
in C++ Exceptions and the Foundation Classes 49
in Type 120
interval (parameter)
in setReplyTimeout 100
invalidArgument
in C++ Exceptions and the Foundation Classes 48
in Type 119
invalidMethodCall
in C++ Exceptions and the Foundation Classes 49
in Type 119
invalidPasswordAttempts
in IccUser class 256
IPMD 46
isAddable
in IccFile class 123
isAddable (continued)
in Writing ESDS records 29
in Writing KSDS records 29
in Writing RRDS records 29

isAPLKeyboard
in IccTerminalData class 238
in Public methods 238

isAPLText
in IccTerminalData class 238
in Public methods 238

isBrowsable
in IccFile class 123

isBTrans
in IccTerminalData class 238

isClassMemoryMgmtOn
in Functions 64
in Icc structure 64

isColor
in IccTerminalData class 239

isCommandSecurityOn
in IccTask class 210

isCommitSupported
in IccTask class 210

isCreated
in IccControl class 104

isDeletable
in IccFile class 123

isDumpAvailable
in IccAbendData class 70

isEDFOn
in Functions 64
in Icc structure 64
in IccResource class 171

isEmptyOnOpen
in IccFile class 123

isErrorSet
in IccSession class 186

isEWA
in IccTerminalData class 239

isExpired
in IccAlarmRequestId class 77

isExtended3270
in IccTerminalData class 239
in Public methods 239

isFamilySubsetEnforcementOn
in Functions 64
in Icc structure 64

isFieldOutline
in IccTerminalData class 239
in Public methods 239

isFMHContained
in IccBuf class 86
in Public methods 86

isGoodMorning
in IccTerminalData class 239
in Public methods 239

isHighlight
in IccTerminalData class 240

isKatakana
in IccTerminalData class 240

isMSRControl
in IccTerminalData class 240

isNoDataSet
in IccSession class 186

isPS
in IccTerminalData class 240

ISR2
in Example of starting transactions 35

isReadable
in IccFile class 124
in Reading ESDS records 28
in Reading KSDS records 28
in Reading RRDS records 28

isReadable method 28

isRecoverable
in IccFile class 124

isResourceSecurityOn
in IccTask class 210

isRestarted
in IccTask class 210

isRouteOptionOn
in IccResource class 171
in Public methods 171

isSignalSet
in IccSession class 186

isSOSI
in IccTerminalData class 240

isStartDataAvailable
in IccTask class 210

issueAbend
in IccTask class 210

issueConfirmation
in IccSession class 187

issueError
in IccSession class 187

issuePrepare
in IccSession class 187

issueSignal
in IccSession class 187

isTextKeyboard
in IccTerminalData class 240
in Public methods 240

isTextPrint
in IccTerminalData class 241
in Public methods 241

isUpdatable
in IccFile class 124

isValidation
in IccTerminalData class 241

item (parameter)
in rewriter Item 219
in write Item 110, 219

itemNum (parameter)
in read Item 218
in rewriter Item 219

ITMP
in Example of starting transactions 35
main (continued)
in Example of time and date services 43
in Header files 6
in main function 265
in Program control 32
in Storage management 58
main function
in Hello World 9
majorCode
in ConditionType 174
manual
in UpdateMode 95
Manual condition handling (noAction)
in CICS conditions 50
in Conditions, errors, and exceptions 50
maxValue
in Range 97
mem (parameter)
in initializeEnvironment 64
memory
in Location 220
message
in IccException class 118
message (parameter)
in Constructor 117
in setInputMessage 233
method
in Foundation Classes-reference 62
Method level
in Conditions, errors, and exceptions 53
in Platform differences 53
methodName
in IccEvent class 116
in IccException class 118
in IccMessage class 153
methodName (parameter)
in Constructor 115, 117, 153
milliSeconds
in IccAbsTime class 74
in IccClock class 92
minorCode
in ConditionType 174
minutes
in IccAbsTime class 74
in IccTime class 243
minutes (parameter)
in Constructor 243, 247, 249
in set 247, 249
Miscellaneous
Example of polymorphic behavior 56
mixed
in Case 235
mode (parameter)
in readNextRecord 133
in readPreviousRecord 134
in readRecord 125
monthOfYear
in Example of time and date services 44
in IccAbsTime class 74
in IccClock class 92
MonthOfYear
in Enumerations 94
in IccClock class 94
msg (parameter)
in clearInputMessage 157
in registerInputMessage 159
in setInputMessage 159
MVS/ESA
in ClassMemoryMgmt 65
in Storage management 58
MVSPost
in WaitPostType 216
MyTempStore
in Automatic condition handling
(callHandleEvent) 51
N
N
in operatingSystem 203
name
in IccResourceId class 175
in IccResourceld class 175
name (parameter)
in Constructor 77, 151, 199, 221, 223, 251, 253, 259
in operator= 151, 199, 221, 223, 251, 253, 259
in setWaitText 212
nameLength
in IccResourceld class 175
NameOpt
in Enumerations 81
in IccBase class 81
netName
in IccTerminal class 227
neutral
in Color 235
new
in Storage management 58
new operator 13
newPassword (parameter)
in changePassword 255
in signon 233
NextTransIdOpt
in Enumerations 235
in IccTerminal class 235
noAccess
in Access 129
noAction
in ActionOnCondition 174
in CICS conditions 50
noCommitOnReturn
in CommitOpt 160
NONCICS
in ASRAKeyType 68
none
in FacilityType 215
noQueue
in AllocateOpt 191
normal
in ReadMode 129
normal (continued)
in SendOpt 191
in TraceOpt 215
NoSpaceOpt
in Enumerations 220
in IccTempStore class 220
noSuspend
in Options 142
notAddable
in Access 129
NOTAPPLIC
in ASRAKeyType 68
in ASRASpaceType 69
in ASRATorageType 69
notBrowsable
in Access 129
notDeletable
in Access 129
notPurgeable
in WaitPurgeability 216
notReadable
in Access 129
notUpdatable
in Access 129
num (parameter)
in operator= 163, 177
in operator!= 163
in operator== 163
in operator<< 88, 227, 228
number
in IccException class 118
in IccJournalId class 143
in IccMessage class 153
in IccRBA class 164
in IccRRN class 178
in IccTask class 211
in Writing RRDS records 29
numEvents (parameter)
in(Constructor 153
in setCustomClassNum 80
numLines (parameter)
in setNewLine 232
numRoutes (parameter)
in setRouteCodes 100

O

obj (parameter)
in Using an object 14
object
creating 13
deleting 14
in GetOpt 66
using 14
object (parameter)
in Constructor 115, 117
in operator delete 80

Object level
in Conditions, errors, and exceptions 52
in Platform differences 52
objectCreationError
in C++ Exceptions and the Foundation Classes 48
in Type 119
offset (parameter)
in cut 85
in dataArea 85
in insert 86
in replace 89
in setCursor 231
onOff (parameter)
in setEDF 64, 172
open
in Status 129
openStatus
in IccFile class 125
operatingSystem
in IccSystem class 203
in Public methods 203
operatingSystemLevel
in IccSystem class 203
operator=
in Example of file control 31
in IccAbsTime class 74
in IccAlarmRequestld class 78
in IccBuf class 86
in IccConvId class 107
in IccDataQueueId class 113
in IccFileld class 131
in IccGroupId class 137
in IccJournalId class 143
in IccJournalTypeld class 145
in IccKey class 148
in IccLockId class 151
in IccPartnerId class 155
in IccProgramId class 161
in IccRBA class 163
in IccRequestId class 167
in IccResourceId class 176
in IccRRN class 177
in IccSysld class 199
in IccTempStoreld class 221
in IccTermld class 223
in IccTimeInterval class 247
in IccTimeOfDay class 249
in IccTPNameId class 251
in IccTransld class 253
in IccUserId class 259
in Protected methods 176
in Public methods 74, 247
in Working with IccResource subclasses 25
operator!=
in IccBuf class 87
in IccKey class 148
in IccRBA class 163
in IccRRN class 177
in Public methods 87
operator+=
in IccBuf class 87
operator==
  in IccBuf class 87
  in IccKey class 148
  in IccRBA class 163
  in IccRRN class 177
operator const char*
  in IccBuf class 86
operator delete
  in IccBase class 80
  in Public methods 80
operator<<
  in IccBuf class 87, 88
  in IccTerminal class 227, 228
  in Working with IccResource subclasses 25
operator new
  in IccBase class 80
opt (parameter)
  in abendCode 67
  in access 121
  in accessMethod 122
  in alternateHeight 237
  in alternateWidth 237
  in ASRAInterrupt 67
  in ASRAKeyType 68
  in ASRAPSW 68
  in ASRARegisters 68
  in ASRASpaceType 69
  in ASRAStrorageType 69
  in className 79
  in defaultHeight 237
  in defaultWidth 238
  in enableStatus 122
  in enterTrace 208, 209
  in graphicCharCodeSet 238
  in graphicCharSetId 238
  in height 226
  in isAddable 123
  in isAPLKeyboard 238
  in isAPLText 238
  in isBrowsable 123
  in isBTrans 239
  in isColor 239
  in isDeletable 123
  in isDumpAvailable 70
  in isEmptyOnOpen 123
  in isEWA 239
  in isExtended3270 239
  in isFieldOutline 239
  in isGoodMorning 239
  in isHighlight 240
  in isKatakana 240
  in isMSRControl 240
  in isPS 240
  in isReadable 124
  in isRecoverable 124
  in isSOSI 240
  in isTextKeyboard 240
  in isTextPrint 241
  in isUpdatable 124
  in isValidable 241
  in keyLength 124
opt (parameter) (continued)
  in keyPosition 124
  in link 158
  in load 158
  in openStatus 125
  in originalAbendCode 70
  in principalSysId 211
  in priority 211
  in programName 70
  in recordFormat 125
  in recordLength 126
  in rewriteItem 219
  in setNextTransId 233
  in type 127
  in userId 212
  in waitExternal 213
  in width 234
  in write 100
  in writeAndGetReply 101
  in writeItem 219
opt1 (parameter)
  in abend 207
opt2 (parameter)
  in abend 207
option (parameter)
  in allocate 184
  in retrieveData 194
  in send 188
  in sendInvite 188, 189
  in sendLast 189
  in state 189
  in stateText 190
  in wait 140
  in writeRecord 141
Options
  in Enumerations 142
  in IccJournal class 142
options (parameter)
  in Constructor 139
opts (parameter)
  in setDumpOpts 211
originalAbendCode
  in IccAbendData class 70
OS/2
  in ClassMemoryMgmt 65
  in Storage management 58
OS/2, CICS
  in Platform differences 52
Other datasets for CICS/ESA
  in Installed contents 6
Output from sample programs
  First Screen 284
  Second Screen 284
overlay
  in IccBuf class 89
overview of Foundation Classes 15
Overview of the foundation classes
  Calling methods on a resource object 20
  Creating a resource object 19
P

P  in operatingSystem 203
PA1 to PA3  in AIDVal 235
packedDecimal  in IccAbsTime class 75
Parameter level  in Conditions, errors, and exceptions 53 in Platform differences 53
parameter passing 58
Parameter passing conventions  in Miscellaneous 58
partnerName (parameter)  in Constructor 155 in operator= 155
password (parameter)  in changePassword 255 in signon 233 in verifyPassword 257
passwordExpiration  in IccUser class 257
PF1 to PF24  in AIDVal 235
pink  in Color 235
PIP (parameter)  in connectProcess 184, 185
PIPList  in IccSession class 187
platform differences  method level 53 object level 52 parameter level 53
Platform differences  in Conditions, errors, and exceptions 52 Method level 53 Object level 52 Parameter level 53
platformError  in Type 119
Platforms  in Enumerations 66 in Icc structure 66
polymorphic behavior 55
Polymorphic Behavior  Example of polymorphic behavior 56 in Miscellaneous 55
popt (parameter)  in setStartOpts 196
prefix (parameter)  in registerPrefix 140 in setPrefix 140
pri (parameter)  in setPriority 211
principalSysId  in IccTask class 211 in Public methods 211
print  in Polymorphic Behavior 55
priority  in IccTask class 211 in Public methods 211
process  in IccSession class 187
profile (parameter)  in Constructor 183
progName (parameter)  in Constructor 157, 161 in operator= 161
program control  example 32 introduction 32
Program control  in Using CICS Services 32
programId  in IccControl class 104 in Method level 53 in Public methods 104
programId (parameter)  in setAbendHandler 105
programName  in IccAbendData class 70 in Public methods 70
programName (parameter)  in setAbendHandler 105
Protected methods  in IccBase class 80 in IccResourceld class 176 operator= 176 setClassName 80 setCustomClassNum 80
ProtectOpt  in Enumerations 198
in IccStartRequestQ class 198
pStorage (parameter)  in freeStorage 202
Public methods  abend 207
abendCode 67
abendData 207
absTime 91
access 121
accessMethod 122
actionOnCondition 169
actionOnConditionAsChar 169
actionsOnConditionsText 170
address 157
AID 225
allocate 184
alternateHeight 237
alternateWidth 237
append 84
applName 201
ASRAInterrupt 67
ASRAKeyType 68
ASRAPSW 68
ASRARegisters 68
ASRASpaceType 69
ASRAStorageType 69
assign 84, 85, 147
Public methods *(continued)*

*beginBrowse* 201
*beginInsert*(VSAM only) 122
*callingProgramId* 103
*cancel* 193
*cancelAbendHandler* 103
*cancelAlarm* 91
*changePassword* 255
*className* 79, 115, 118, 153
*classType* 79, 115, 118
*clear* 109, 170, 217, 225
*clearData* 193
*clearInputMessage* 157
*clearPrefix* 139
*commArea* 103
*commitUOW* 207
*completeLength* 147
*condition* 115, 170
*conditionText* 116, 170
*connectProcess* 184
*console* 104
*converse* 185
*convId* 185
*cursor* 225
*customClassNum* 79
*cut* 85
*data* 194, 225
*dataArea* 85
*dataAreaLength* 85
*dataAreaOwner* 85
*dataAreaType* 85
*dataLength* 85
*date* 73, 91
*dateFormat* 202
*dayOfMonth* 73, 92
*dayOfWeek* 74, 92
*daysSince1900* 74, 92
*daysUntilPasswordExpires* 256
*defaultHeight* 237
*defaultWidth* 238
*delay* 208
*deleteLockedRecord* 122
*deleteRecord* 122
*dump* 208
*empty* 109, 217
*enableStatus* 122
*endBrowse* 202
*endInsert*(VSAM only) 123
*enterTrace* 208
*entryPoint* 158
*erase* 226
*errorCode* 185
*ESMReason* 256
*ESMResponse* 256
*extractProcess* 185
*facilityType* 209
*flush* 186
*free* 186
*freeKeyboard* 226
*freeStorage* 202, 209
*get* 109, 170, 186, 218, 226

Public methods *(continued)*

*getFile* 202
*getNextFile* 203
*getStorage* 203, 209
*graphicCharCodeSet* 238
*graphicCharSetId* 238
*groupId* 256
*handleEvent* 171
*height* 226
*hours* 74, 243
*id* 171
*in IccAbendData class* 67
*in IccAbsTime class* 73
*in IccAlarmRequestId class* 77
*in IccBase class* 79
*in IccBuf class* 84
*in IccClock class* 91
*in IccConsole class* 99
*in IccControl class* 103
*in IccConvId class* 107
*in IccDataQueue class* 109
*in IccDataQueueEnd class* 113
*in IccEvent class* 115
*in IccException class* 118
*in IccFile class* 121
*in IccField class* 131
*in IccFileIterator class* 133
*in IccGroupId class* 137
*in IccJournal class* 139
*in IccJournalId class* 143
*in IccJournalTypeld class* 145
*in IccKey class* 147
*in IccLockId class* 151
*in IccMessage class* 153
*in IccPartnerId class* 155
*in IccProgram class* 157
*in IccProgramId class* 161
*in IccRBA class* 163
*in IccRecordIndex class* 165
*in IccRequestld class* 167
*in IccResource class* 169
*in IccResourceId class* 175
*in IccRRN class* 177
*in IccSemaphore class* 179
*in IccSession class* 184
*in IccStartRequestQ class* 193
*in IccSysld class* 199
*in IccSystem class* 201
*in IccTask class* 207
*in IccTempStore class* 217
*in IccTempStoreld class* 221
*in IccTermId class* 223
*in IccTerminal class* 225
*in IccTerminalData class* 237
*in IccTime class* 243
*in IccTimeInterval class* 247
*in IccTimeOfDay class* 249
*in IccTPNameId class* 251
*in IccTransId class* 253
*in IccUser class* 255
*in IccUserId class* 259
Public methods (continued)

initData 104
inputCursor 226
insert 86
instance 70, 99, 104, 194, 203, 210, 226
invalidPasswordAttempts 256
isAddable 123
isAPLKeyboard 238
isAPLText 238
isBrowsable 123
isBTrans 238
isColor 239
isCommandSecurityOn 210
isCommitSupported 210
isCreated 104
isDeletable 123
isDumpAvailable 70
isEDFOn 171
isEmptyOnOpen 123
isErrorSet 186
isEWA 239
isExpired 77
isExtended3270 239
isFieldOutline 239
isFMHContained 86
isGoodMorning 239
isHighlight 240
isKatakana 240
isMSRControl 240
isNoDataSet 186
isPS 240
isReadable 124
isRecoverable 124
isResourceSecurityOn 210
isRestarted 210
isRouteOptionOn 171
isSignalSet 186
isSOSI 240
isStartDataAvailable 210
issueAbend 186
issueConfirmation 187
issueError 187
issuePrepare 187
issueSignal 187
isTextKeyboard 240
isTextPrint 241
isUpdatable 124
isValidation 241
journalTypeId 140
keyLength 124
keyPosition 124
kind 147
language 256
lastPasswordChange 256
lastUseTime 256
length 158, 165
lifeTime 179
line 226
link 158
load 158
lock 180

Public methods (continued)

message 118
methodName 116, 118, 153
milliSeconds 74, 92
minutes 74, 243
monthOfYear 74, 92
name 171, 175
nameLength 175
netName 227
number 118, 143, 153, 164, 178, 211
numberOfItems 218
openStatus 125
operatingSystem 203
operatingSystemLevel 203
operator 74, 78, 86, 107, 113, 131, 137, 143, 145,
148, 151, 155, 161, 163, 167, 177, 199, 221, 223,
247, 249, 251, 253, 259
operator!= 87, 148, 163, 177
operator+= 87
operator== 87, 148, 163, 177
operator const char* 86
operator delete 80
operator new 80
operator << 87, 88, 227, 228
operator new 80
originalAbendCode 70
overlay 89
packedDecimal 75
passwordExpiration 257
PIPLList 187
principalSysId 211
priority 211
process 187
programId 104
programName 70
put 99, 110, 140, 171, 188, 218, 228
queueName 194
readItem 110, 218
readNextItem 218
readNextRecord 133
readPreviousRecord 134
readRecord 125
receive 188, 228
receive3270Data 229
recordFormat 125
recordIndex 126
recordLength 126
registerData 194
registerInputMessage 159
registerPrefix 140
registerRecordIndex 126
release 204
releaseText 204
replace 89
replyTimeout 99
reset 134, 194
resetAbendHandler 104
resetRouteCodes 100
retrieveData 194
returnProgramId 105
returnTermId 195
returnTransId 195
Public methods (continued)
rewriteltem 219
rewriteRecord 126
rollBackUOW 211
routeOption 172
run 105
seconds 75, 243
send 188, 229
send3270 230
sendInvite 188, 189
sendLast 189
sendLine 230, 231
session 105
set 247, 249
setAbendHandler 105
setAccess 127
setActionOnAnyCondition 172
setActionOnCondition 172
setActionsOnConditions 172
setAlarm 93
setAllRouteCodes 100
setColor 231
setCursor 231, 232
setData 195
setDataLength 89
setDumpOpts 211
setEDF 172
setEmptyOnOpen 127
setFMHContained 89
setHighlight 232
setInputMessage 159
setJournalTypeeld 140
setKind 148
setLanguage 257
setLine 232
setNewLine 232
setNextCommArea 232
setNextInputMessage 233
setNextTransId 233
setPrefix 140
setPriority 211
setQueueName 195
setReplyTimeout 100
setReturnTermld 195
setReturnTransId 196
setRouteCodes 100
setRouteOption 173
setStartOpts 196
setStatus 127
setTimerECA 78
setWaitText 212
signoff 233
signon 233
start 196
startRequestQ 105
startType 212
state 189
stateText 190
summary 116, 118, 153
suspend 212
syncLevel 190

Public methods (continued)
syssel 204
system 105
task 106
terminal 106
text 154
time 75, 93
timeInHours 75, 243
timeInMinutes 75, 244
timeInSeconds 75, 244
timerECA 78
transId 212
triggerDataQueueId 212
tryLock 180
type 118, 127, 165, 180, 244
typeText 119
unlock 180
unlockRecord 128
update 93
userId 212
value 148
verifyPassword 257
wait 140
waitExternal 213
waitForAID 234
waitOnAlarm 213
width 234
workArea 204, 213, 234
write 100
writeAndGetReply 101
writeItem 110, 219
writeRecord 128, 141
year 75, 93
purgeable
in WaitPurgeability 216
put
in Example of polymorphic behavior 57
in iccConsole class 99
in iccDataQueue class 110
in iccJournal class 140
in iccResource class 171
in iccSession class 188
in iccTempStore class 218
in iccTerminal class 228
in Polymorphic Behavior 56

Q
queue
in AllocateOpt 191
in NextTransIdOpt 235
queueName
in Accessing start data 34
in iccStartRequestQ class 194
queueName (parameter)
in Constructor 109, 113
in operator= 113
in setQueueName 195
rAbendTask
  in HandleEventReturnOpt 174
Range
  in Enumerations 97
  in IccCondition structure 97
RBA 27
rba (parameter)
  in operator== 163
  in operator!= 163
rContinue
  in HandleEventReturnOpt 174
readable
  in Access 129
reading data 37
Reading data
  in Transient Data 37
  in Using CICS Services 37
Reading ESDS records
  in File control 28
  in Reading records 28
reading items 39
Reading items
  in Temporary storage 39
  in Using CICS Services 39
Reading KSDS records
  in File control 28
  in Reading records 28
Reading records
  in File control 27
  in Using CICS Services 27
Reading ESDS records 28
Reading KSDS records 28
Reading RRDS records 28
Reading RRDS records
  in File control 28
  in Reading records 28
readItem
  in Example of Temporary Storage 41
  in IccDataQueue class 110
  in IccTempStore class 218
  in Reading data 37
  in Reading items 39
  in Scope of data in IccBuf reference returned from 'read' methods 59
  in Temporary storage 39
  in Transient Data 37
  in Working with IccResource subclasses 25
ReadMode
  in Enumerations 129
  in IccFiles class 129
readNextItem
  in IccTempStore class 218
  in Scope of data in IccBuf reference returned from 'read' methods 59
  in Temporary storage 39
readNextRecord
  in Browsing records 30
  in IccFileIterator class 133
  in Public methods 133
readNextRecord method 30
READONLY
  in ASRASStorageType 69
readPreviousRecord 30
  in Browsing records 30
  in IccFileIterator class 134
readRecord
  in C++ Exceptions and the Foundation Classes 49
  in Deleting locked records 30
  in IccFile class 125
  in Reading records 28
  in Updating records 29
readRecord method 28
receive
  in IccSession class 188
  in IccTerminal class 228
  in Receiving data from a terminal 42
receive3270data
  in Receiving data from a terminal 42
receive3270Data
  in IccTerminal class 229
  in Public methods 229
receiving data from a terminal 42
Receiving data from a terminal
  in Terminal control 42
  in Using CICS Services 42
record (parameter)
  in writeRecord 141
recordFormat
  in IccFile class 125
  in Reading ESDS records 28
  in Reading RRDS records 28
  in Writing ESDS records 29
  in Writing RRDS records 29
recordFormat method 28
recordIndex
  in IccFile class 126
  in Reading ESDS records 28
  in Reading KSDS records 28
  in Reading RRDS records 28
  in Writing ESDS records 29
  in Writing KSDS records 29
  in Writing RRDS records 29
recordIndex method 28
recordLength
  in IccFile class 126
  in Reading ESDS records 28
  in Reading KSDS records 28
  in Reading RRDS records 28
  in Writing ESDS records 29
  in Writing KSDS records 29
  in Writing RRDS records 29
recordLength method 28
red
  in Color 235
registerData 193
  in Example of starting transactions 35
  in IccStartRequestQ class 194
  in Starting transactions 34
registerInputMessage 157
  in IccTerminal class 159
registerPrefix
  in IccJournal class 140
  in Public methods 140
registerRecordIndex 28
  in IccFile class 126
  in Reading ESDS records 28
  in Reading KSRS records 28
  in Writing ESDS records 29
  in Writing KSRS records 29
  in Writing records 28
  in Writing RRDS records 29
registerRecordIndex method 28
relative byte address 27
relative record number 27
release
  in IccSystem class 204
releaseAtTaskEnd
  in LoadOpt 160
releaseText
  in IccSystem class 204
remoteTermId
  in Example of starting transactions 35
replace
  in IccBuf class 89
  in IccBuf constructors 24
replyTimeout
  in IccConsole class 99
req
  in Example of starting transactions 36
req1
  in Example of starting transactions 35
req2
  in Example of starting transactions 35
reqestName (parameter)
  in Constructor 167
reqld (parameter)
  in cancel 193
  in cancelAlarm 91
  in delay 208
  in setAlarm 93
  in start 197
requestName (parameter)
  in Constructor 167
  in operator= 78, 167
requestNum (parameter)
  in wait 140
reset
  in Browsing records 30
  in IccFileIterator class 134
  in IccStartRequestQ class 194
resetAbendHandler
  in IccSystem class 104
resetRouteCodes
  in IccConsole class 100
  in Public methods 100
resId (parameter)
  in beginBrowse 201
resName (parameter)
  in beginBrowse 201, 202
  in Constructor 175
resource (parameter)
  in beginBrowse 201
  in Constructor 179
  in endBrowse 202
  in enterTrace 208, 209
resource class 17
Resource classes
  in Overview of the foundation classes 17
resource identification class 16
Resource identification classes
  in Overview of the foundation classes 16
resource object
  creating 19
ResourceType
  in Enumerations 205
  in IccSystem class 205
respectAbendHandler
  in AbendHandlerOpt 214
retrieveData
  in Accessing start data 34
  in IccStartRequestQ class 193, 194
  in Mapping EXEC CICS calls to Foundation Class
  methods 275
RetrieveOpt
  in Enumerations 198
  in IccStartRequestQ class 198
return
  in Mapping EXEC CICS calls to Foundation Class
  methods 275
returnCondition
  in NoSpaceOpt 220
returnProgramId
  in IccControl class 105
  in Public methods 105
returnTermId
  in Accessing start data 34
  in IccStartRequestQ class 195
returnToCICS
  in Functions 64
  in Icc structure 64
returnTransId
  in Accessing start data 34
  in IccStartRequestQ class 195
reverse
  in Highlight 235
rewritetem
  in Example of Temporary Storage 41
  in IccTempStore class 219
  in Temporary storage 39
  in Updating items 40
  in Writing items 39
rewriteRecord
  in IccFile class 126
  in Updating records 29
rewriteRecord method 29
rewriting records 29
rollBackUOW
  in IccTask class 211
routeOption
  in IccResource class 172
row (parameter)
in send 229
in setCursor 232
RRDS file
in File control 27
RRN 27
rrn (parameter)
in operator== 177
in operator!= 177
in operator== 177
rThrowException
in HandleEventReturnOpt 174
run
in Base classes 15
in C++ Exceptions and the Foundation Classes 48
in Example of file control 30, 32
in Example of managing transient data 38, 39
in Example of polymorphic behavior 56
in Example of starting transactions 35
in Example of Temporary Storage 40, 41
in Example of terminal control 42, 43
in Example of time and date services 44
in Hello World 10
in IccControl class 103, 105
in main function 265, 266
in Mapping EXEC CICS calls to Foundation Class methods 275
in Program control 32
run method
in Hello World 9
Running "Hello World" on your CICS server
Expected Output from "Hello World" 10
in Hello World 10
Running the sample applications. 6

S
sample source 6
Sample source code
in Installed contents 6
Location 6
scope of data 59
Scope of data in IccBuf reference returned from 'read' methods
in Miscellaneous 59
scope of references 59
search (parameter)
in Constructor 133
in reset 134
SearchCriterion
in Enumerations 129
in IccFile class 129
Second Screen
in ICC$PRG1 (IPR1) 284
in Output from sample programs 284
seconds
in IccAbsTime class 75
in IccTime class 243
seconds (parameter)
in Constructor 243, 247, 249
in set 247, 249
seconds (parameter) (continued)
in setReplyTimeout 100
send
in Example of terminal control 42
in Hello World 10
in IccSession class 188
in IccTerminal class 229
in Sending data to a terminal 41
send (parameter)
in converse 185
in put 99
in send 188
in sendInvite 188
in sendLast 189
in write 100
in writeAndGetReply 101
send3270
in IccTerminal class 230
sending data to a terminal 41
Sending data to a terminal
in Terminal control 41
in Using CICS Services 41
sendInvite
in IccSession class 188, 189
sendLast
in IccSession class 189
sendLine
in Example of file control 31
in Example of terminal control 42
in IccTerminal class 230, 231
in Sending data to a terminal 41
SendOpt
in Enumerations 191
in IccSession class 191
sequential reading of files 30
session
in FacilityType 215
in IccControl class 105
set
in IccTimeInterval class 247
in IccTimeOfDay class 249
set...
in Sending data to a terminal 41
set (parameter)
in boolText 63
setAbendHandler
in IccControl class 105
setAccess
in IccFile class 127
setActionOnAnyCondition
in IccResource class 172
setActionOnCondition
in IccResource class 172
setActionsOnConditions
in IccResource class 172
setAlarm
in IccAlarmRequestId class 77
in IccClock class 93
setAllRouteCodes
in IccConsole class 100
setClassName
  in IccBase class 80
  in Protected methods 80
setColor
  in Example of terminal control 43
  in IccTerminal class 231
setCursor
  in IccTerminal class 231, 232
setCustomClassNum
  in IccBase class 80
  in Protected methods 80
setData 193
  in IccStartRequestQ class 195
  in Starting transactions 34
setDataLength
  in IccBuf class 89
setDumpOpts
  in IccTask class 211
setEDF
  in Functions 64
  in Icc structure 64
  in IccResource class 172
setEmptyOnOpen
  in IccFile class 127
  in Public methods 127
setFMHContained
  in IccBuf class 89
  in Public methods 89
setHighlight
  in Example of terminal control 43
  in IccTerminal class 232
setInputMessage 157
  in IccProgram class 159
  in Public methods 159
setJournalTypeId
  in IccJournal class 140
setKind
  in Example of file control 31
  in IccKey class 148
setLanguage
  in IccUser class 257
setLine
  in IccTerminal class 232
setNewLine
  in IccTerminal class 232
setNextCommArea
  in IccTerminal class 232
  in Public methods 232
setNextInputMessage
  in IccTerminal class 233
setNextTransId
  in IccTerminal class 233
setPrefix
  in IccJournal class 140
setPriority
  in IccTask class 211
  in Public methods 211
setQueueName
  in Example of starting transactions 36
  in IccStartRequestQ class 195
  in Starting transactions 34
setReplyTimeout
  in IccConsole class 100
setReturnTermId
  in Example of starting transactions 35
  in IccStartRequestQ class 195
  in Starting transactions 34
setReturnTransId
  in Example of starting transactions 35
  in IccStartRequestQ class 196
  in Starting transactions 34
setRouteCodes
  in IccConsole class 100
setRouteOption
  in Example of starting transactions 35, 37
  in IccResource class 173
  in Program control 33
  in Public methods 173
setStartOpts
  in IccStartRequestQ class 196
setStatus
  in IccFile class 127
setTimerECA
  in IccAlarmRequestId class 78
setWaitText
  in IccTask class 212
Severe error handling (abendTask)
  in CICS conditions 52
  in Conditions, errors, and exceptions 52
SeverityOpt
  in Enumerations 102
  in IccConsole class 102
signoff
  in IccTerminal class 233
signon
  in IccTerminal class 233
  in Public methods 233
singleton class 20
Singleton classes
  in Creating a resource object 20
  in Using CICS resources 20
size (parameter)
  in getStorage 203, 209
  in operator new 80
start
  in Example of starting transactions 36
  in IccRequestId class 167
  in IccStartRequestQ class 193, 196
  in Mapping EXEC CICS calls to Foundation Class methods 275
  in Parameter passing conventions 58
  in Starting transactions 34
Starting transactions
  in Starting transactions asynchronously 34
  in Using CICS Services 34
starting transactions asynchronously 34
Starting transactions asynchronously
  Accessing start data 34
  Cancelling unexpired start requests 34
  Example of starting transactions 34
  in Using CICS Services 34
  Starting transactions 34
startIO
   in Options 142
startRequest
   in StartType 215
startRequestQ
   in Example of starting transactions 35, 36
   in IccControl class 105
startType
   in Example of starting transactions 36
   in IccTask class 212
StartType
   in Enumerations 215
   in IccTask class 215
state
   in IccSession class 189
StateOpt
   in Enumerations 191
   in IccSession class 191
stateText
   in IccSession class 190
Status
   in Enumerations 129
   in IccFile class 129
status (parameter)
   in setStatus 127
Storage management
   in Miscellaneous 57
StorageOpts
   in Enumerations 215
   in IccTask class 215
storageOpts (parameter)
   in getStorage 203, 209
storeName (parameter)
   in Constructor 217
SUBSPACE
   in ASRASpaceType 69
summary
   in IccEvent class 116
   in IccException class 118
   in IccMessage class 153
support classes 18
Support Classes
   in Overview of the foundation classes 18
suppressDump
   in AbendDumpOpt 214
suspend
   in IccTask class 212
   in NoSpaceOpt 220
symbolic debuggers 46
Symbolic Debuggers
   in Compiling, executing, and debugging 46
   in Debugging Programs 46
synchronous
   in Options 142
syncLevel
   in IccSession class 190
SyncLevel
   in Enumerations 191
   in IccSession class 191
sysId
   in IccSystem class 204
sysId (parameter)
   in Constructor 183
   in setRouteOption 173
sysName (parameter)
   in Constructor 183
   in setRouteOption 173
system
   in IccControl class 105

T

   task
      in IccControl class 106
      in LifeTime 181
temporary storage
   deleting items 40
      example 40
      introduction 39
      reading items 39
      updating items 40
      Writing items 39
Temporary storage
   Deleting items 40
   Example of Temporary Storage 40
   in Using CICS Services 39
   Reading items 39
   Updating items 40
   Writing items 39
termId (parameter)
   in setReturnTermId 195
   in start 197
terminal
   finding out about 42
   in FacilityType 215
   in Hello World 9
   in IccControl class 106
   receiving data from 42
   sending data to 41
terminal control
   example 42
   finding out information 42
   introduction 41
   receiving data 42
   sending data 41
Terminal control
   Example of terminal control 42
   Finding out information about a terminal 42
   in Using CICS Services 41
   Receiving data from a terminal 42
   Sending data to a terminal 41
terminalInput
   in StartType 215
termName (parameter)
   in setReturnTermId 195
Test
   in C++ Exceptions and the Foundation Classes 47, 48
test (parameter)
   in boolText 63
text
   in IccMessage class 154
Tracing a Foundation Class Program
Activating the trace output 46
in Compiling, executing, and debugging 46
in Debugging Programs 46

transId
in IccTask class 212
transId (parameter)
in setNextTransId 233
transId (parameter)
in cancel 193
in connectProcess 184
in link 158
in setNextTransId 233
in setReturnTransId 196
in start 197
transient data
deleting queues 38
example 38
introduction 37
reading data 37
Writing data 38

Transient Data
Deleting queues 38
Example of managing transient data 38
in Using CICS Services 37
Reading data 37
Writing data 38

transName (parameter)
in setReturnTransId 196
triggerDataQueueId
in IccTask class 212
trueFalse (parameter)
in setEmptyOnOpen 127
try
in C++ Exceptions and the Foundation Classes 47, 48
in Exception handling (throwException) 51, 52
in main function 266
tryLock
in IccSemaphore class 180
tryNumber
in C++ Exceptions and the Foundation Classes 47, 48
type
in C++ Exceptions and the Foundation Classes 48
in IccException class 118
in IccFile class 127
in IccRecordIndex class 165
in IccSemaphore class 180
in IccTime class 244
Type
in Enumerations 119, 166, 244
in IccException class 119
in IccRecordIndex class 166
in IccTime class 244
type (parameter)
in condition 115, 170
in Constructor 79, 83, 84, 165, 175, 179
in waitExternal 213
typeText
in IccException class 119
Using CICS resources
   Calling methods on a resource object 20
   Creating a resource object 19
   in Overview of the foundation classes 19
   Singleton classes 20

Using CICS Services
   Accessing start data 34
   Browsing records 30
   Cancelling unexpired start requests 34
   Deleting items 40
   Deleting queues 38
   Deleting records 29
   Example of file control 30
   Example of managing transient data 38
   Example of starting transactions 34
   Example of Temporary Storage 40
   Example of terminal control 42
   Example of time and date services 43
   Finding out information about a terminal 42
   Reading data 37
   Reading items 39
   Reading records 27
   Receiving data from a terminal 42
   Sending data to a terminal 41
   Starting transactions 34
   Updating items 40
   Updating records 29
   Writing data 38
   Writing items 39
   Writing records 28

V

value
   in IccKey class 148
value (parameter)
   in operator= 148
variable (parameter)
   in Foundation Classes—reference 62
verifyPassword
   in IccUser class 257
   in Public methods 257
virtual
   in Glossary 289
VSAM 27

W

wait
   in IccJournal class 140
   in SendOpt 191
waitExternal
   in IccTask class 213
waitForAID
   in Example of terminal control 43
   in IccTerminal class 234
waitOnAlarm
   in IccAlarmRequestId class 77
   in IccTask class 213
WaitPostType
   in Enumerations 216
This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

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

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

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

The following paragraph does not apply in the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This publication 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.
The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

**Trademarks**

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

<table>
<thead>
<tr>
<th>AIX</th>
<th>AT</th>
<th>C Set++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common User Access</td>
<td>CICS</td>
<td>CICS OS/2</td>
</tr>
<tr>
<td>CICS TS</td>
<td>DB/2</td>
<td>IBM</td>
</tr>
<tr>
<td>Language Environment</td>
<td>OS/390</td>
<td>OS/2</td>
</tr>
<tr>
<td>VisualAge</td>
<td>VTAM</td>
<td></td>
</tr>
</tbody>
</table>

Other company, product, and service names may be trademarks or service marks of others.
Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To ask questions, make comments about the functions of IBM products or systems, or to request additional publications, contact your IBM representative or your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

• By mail, to this address:
  
  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER
  Hampshire
  SO21 2JN
  United Kingdom

• By fax:
  – From outside the U.K., after your international access code use 44–1962–842327
  – From within the U.K., use 01962–842327

• Electronically, use the appropriate network ID:
  – IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  – IBMLink™: HURSLEY(IDRCF)
  – Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

• The publication title and order number
• The topic to which your comment applies
• Your name and address/telephone number/fax number/network ID.
Spine information:

IBM  CICS Transaction Server  C++ OO Class Libraries