

CICS Transaction Server for z/OS



C++ OO Class Libraries

Version 3 Release 1

CICS Transaction Server for z/OS



C++ OO Class Libraries

Version 3 Release 1

Note!

Before using this information and the product it supports, be sure to read the general information under Notices” on page 345.

This edition applies to Version 3 Release 1 of CICS Transaction Server for z/OS, program number 5655-M15, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

© Copyright IBM Corporation 1989, 2010.

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

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 13

Chapter 4. C++ Objects	15
Creating an object.	15
Using an object.	16
Deleting an object.	16
Chapter 5. Overview of the foundation classes	17
Base classes	17
Resource identification classes	18
Resource classes	19
Support Classes	20
Using CICS resources	21
Creating a resource object	21
Calling methods on a resource object	22
Chapter 6. Buffer objects	25
IccBuf class	25
Data area ownership.	25
Data area extensibility	25
IccBuf constructors	25
IccBuf methods.	26
Working with IccResource subclasses	26
Chapter 7. Using CICS Services	29
File control	29
Reading records	29

Writing records	30
Updating records	31
Deleting records	31
Browsing records	32
Example of file control	32
Program control	34
Starting transactions asynchronously	36
Starting transactions	36
Accessing start data	36
Cancelling unexpired start requests	36
Example of starting transactions	36
Transient Data	39
Reading data	39
Writing data	40
Deleting queues	40
Example of managing transient data	40
Temporary storage	41
Reading items	41
Writing items	41
Updating items	42
Deleting items	42
Example of Temporary Storage	42
Terminal control	43
Sending data to a terminal	43
Receiving data from a terminal	44
Finding out information about a terminal	44
Example of terminal control	44
Time and date services	45
Example of time and date services	45
Chapter 8. Compiling, executing, and debugging	49
Compiling Programs	49
Executing Programs	49
Debugging Programs	49
Symbolic Debuggers	50
Tracing a Foundation Class Program	50
Execution Diagnostic Facility	50
Chapter 9. Conditions, errors, and exceptions	51
Foundation Class Abend codes	51
C++ Exceptions and the Foundation Classes	51
CICS conditions	53
Manual condition handling (noAction)	54
Automatic condition handling (callHandleEvent)	54
Exception handling (throwException)	55
Severe error handling (abendTask)	56
Platform differences	56
Object level	56
Method level	57
Parameter level	57
Chapter 10. Miscellaneous	59
Polymorphic Behavior	59
Example of polymorphic behavior	60
Storage management	61
Parameter passing conventions	62

Part 3. Foundation Classes—reference 65

Chapter 11. lcc structure. 67

Functions 67

- boolText 67
- catchException 67
- conditionText. 67
- initializeEnvironment 67
- isClassMemoryMgmtOn 68
- isEDFOn 68
- isFamilySubsetEnforcementOn 68
- returnToCICS 68
- setEDF. 68
- unknownException 69

Enumerations 69

- Bool 69
- BoolSet 69
- ClassMemoryMgmt 69
- FamilySubset 70
- GetOpt 70
- Platforms 70

Chapter 12. lccAbendData class. 71

lccAbendData constructor (protected) 71

- Constructor 71

Public methods 71

- abendCode 71
- ASRAInterrupt 71
- ASRAKeyType 72
- ASRAPSW 72
- ASRARegisters 72
- ASRASpaceType 73
- ASRAStorageType 73
- instance 74
- isDumpAvailable 74
- originalAbendCode 74
- programName 74

Inherited public methods 74

Inherited protected methods 75

Chapter 13. lccAbsTime class. 77

lccAbsTime constructor 77

- Constructor (1) 77
- Constructor (2) 77

Public methods 77

- date 77
- dayOfMonth 77
- dayOfWeek 78
- daysSince1900 78
- hours 78
- milliSeconds 78
- minutes 78
- monthOfYear. 78
- operator= 79

packedDecimal	79
seconds	79
time	79
timeInHours	79
timeInMinutes	79
timeInSeconds	79
year	79
Inherited public methods	80
Inherited protected methods	80
Chapter 14. IccAlarmRequestId class	81
IccAlarmRequestId constructors	81
Constructor (1)	81
Constructor (2)	81
Constructor (3)	81
Public methods	81
isExpired	81
operator= (1)	82
operator= (2)	82
operator= (3)	82
setTimerECA	82
timerECA	82
Inherited public methods	82
Inherited protected methods	82
Chapter 15. IccBase class	85
IccBase constructor (protected)	85
Constructor	85
Public methods	85
classType	85
className	85
customClassNum	86
operator delete	86
operator new	86
Protected methods	86
setClassName	86
setCustomClassNum	86
Enumerations	87
ClassType	87
NameOpt	87
Chapter 16. IccBuf class	89
IccBuf constructors	89
Constructor (1)	89
Constructor (2)	89
Constructor (3)	89
Constructor (4)	90
Public methods	90
append (1)	90
append (2)	90
assign (1)	90
assign (2)	91
cut	91
dataArea	91
dataAreaLength	91
dataAreaOwner	91

dataAreaType	91
dataLength	92
insert	92
isFMHContained	92
operator const char*	92
operator= (1)	92
operator= (2)	92
operator+= (1)	93
operator+= (2)	93
operator==	93
operator!=.	93
operator<< (1)	93
operator<< (2)	94
operator<< (3)	94
operator<< (4)	94
operator<< (5)	94
operator<< (6)	94
operator<< (7)	94
operator<< (8)	94
operator<< (9)	94
operator<< (10).	94
operator<< (11).	95
operator<< (12).	95
operator<< (13).	95
operator<< (14).	95
operator<< (15).	95
overlay	95
replace	95
setDataLength	96
setFMHContained.	96
Inherited public methods	96
Inherited protected methods	96
Enumerations	97
DataAreaOwner	97
DataAreaType	97
Chapter 17. lccClock class	99
lccClock constructor	99
Constructor	99
Public methods	99
absTime	99
cancelAlarm	99
date	99
dayOfMonth	100
dayOfWeek.	100
daysSince1900	100
milliSeconds	100
monthOfYear	100
setAlarm.	101
time	101
update	101
year	101
Inherited public methods	101
Inherited protected methods	102
Enumerations	102
DateFormat.	102

DayOfWeek	102
MonthOfYear	102
UpdateMode	103
Chapter 18. IccCondition structure	105
Enumerations	105
Codes.	105
Range	105
Chapter 19. IccConsole class	107
IccConsole constructor (protected)	107
Constructor.	107
Public methods	107
instance	107
put	107
replyTimeout	107
resetRouteCodes	108
setAllRouteCodes	108
setReplyTimeout (1)	108
setReplyTimeout (2)	108
setRouteCodes	108
write	108
writeAndGetReply	109
Inherited public methods	109
Inherited protected methods	109
Enumerations	110
SeverityOpt.	110
Chapter 20. IccControl class	111
IccControl constructor (protected).	111
Constructor	111
Public methods	111
callingProgramId	111
cancelAbendHandler	111
commArea	111
console	112
initData	112
instance	112
isCreated	112
programId	112
resetAbendHandler	112
returnProgramId	113
run	113
session	113
setAbendHandler (1)	113
setAbendHandler (2)	113
startRequestQ.	113
system	113
task	114
terminal	114
Inherited public methods	114
Inherited protected methods.	114
Chapter 21. IccConvId class	115
IccConvId constructors	115
Constructor (1)	115

Constructor (2)	115
Public methods	115
operator= (1)	115
operator= (2)	115
Inherited public methods	115
Inherited protected methods	116
Chapter 22. IccDataQueue class	117
IccDataQueue constructors	117
Constructor (1)	117
Constructor (2)	117
Public methods	117
clear	117
empty	117
get	117
put	118
readItem	118
writeItem (1)	118
writeItem (2)	118
Inherited public methods	118
Inherited protected methods	119
Chapter 23. IccDataQueueIeld class	121
IccDataQueueIeld constructors	121
Constructor (1)	121
Constructor (2)	121
Public methods	121
operator= (1)	121
operator= (2)	121
Inherited public methods	121
Inherited protected methods	122
Chapter 24. IccEvent class	123
IccEvent constructor	123
Constructor	123
Public methods	123
className	123
classType	123
condition	123
conditionText	124
methodName	124
summary	124
Inherited public methods	124
Inherited protected methods	124
Chapter 25. IccException class	125
IccException constructor	125
Constructor	125
Public methods	126
className	126
classType	126
message	126
methodName	126
number	126
summary	126
type	127

typeText	127
Inherited public methods	127
Inherited protected methods	127
Enumerations	127
Type	127
Chapter 26. IccFile class	129
IccFile constructors	129
Constructor (1)	129
Constructor (2)	129
Public methods	129
access	129
accessMethod	130
beginInsert(VSAM only)	130
deleteLockedRecord	130
deleteRecord	130
enableStatus	131
endInsert(VSAM only)	131
isAddable	131
isBrowsable	131
isDeletable	131
isEmptyOnOpen	132
isReadable	132
isRecoverable	132
isUpdatable	132
keyLength	132
keyPosition	133
openStatus	133
readRecord	133
recordFormat	134
recordIndex	134
recordLength	134
registerRecordIndex	134
rewriteRecord	135
setAccess	135
setEmptyOnOpen	135
setStatus	135
type	136
unlockRecord	136
writeRecord	136
Inherited public methods	136
Inherited protected methods	137
Enumerations	137
Access	137
ReadMode	137
SearchCriterion	138
Status	138
Chapter 27. IccFileId class	139
IccFileId constructors	139
Constructor (1)	139
Constructor (2)	139
Public methods	139
operator= (1)	139
operator= (2)	139
Inherited public methods	139

Inherited protected methods	140
Chapter 28. IccFileIterator class	141
IccFileIterator constructor	141
Constructor	141
Public methods	141
readNextRecord	141
readPreviousRecord	142
reset	142
Inherited public methods	142
Inherited protected methods	143
Chapter 29. IccGroupId class	145
IccGroupId constructors	145
Constructor (1)	145
Constructor (2)	145
Public methods	145
operator= (1)	145
operator= (2)	145
Inherited public methods	145
Inherited protected methods	146
Chapter 30. IccJournal class	147
IccJournal constructors	147
Constructor (1)	147
Constructor (2)	147
Public methods	147
clearPrefix	147
journalTypeId	148
put	148
registerPrefix	148
setJournalTypeId (1)	148
setJournalTypeId (2)	148
setPrefix (1)	148
setPrefix (2)	148
wait	148
writeRecord (1)	149
writeRecord (2)	149
Inherited public methods	149
Inherited protected methods	150
Enumerations	150
Options	150
Chapter 31. IccJournalId class	151
IccJournalId constructors	151
Constructor (1)	151
Constructor (2)	151
Public methods	151
number	151
operator= (1)	151
operator= (2)	151
Inherited public methods	152
Inherited protected methods	152
Chapter 32. IccJournalTypeId class	153
IccJournalTypeId constructors	153

Constructor (1)	153
Constructor (2)	153
Public methods	153
operator= (1)	153
operator= (2)	153
Inherited public methods	153
Inherited protected methods	154
Chapter 33. IccKey class	155
IccKey constructors	155
Constructor (1)	155
Constructor (2)	155
Constructor (3)	155
Public methods	155
assign	155
completeLength	155
kind	155
operator= (1)	156
operator= (2)	156
operator= (3)	156
operator== (1)	156
operator== (2)	156
operator== (3)	156
operator!= (1)	156
operator!= (2)	156
operator!= (3)	156
setKind	156
value	157
Inherited public methods	157
Inherited protected methods	157
Enumerations	157
Kind	157
Chapter 34. IccLockId class	159
IccLockId constructors	159
Constructor (1)	159
Constructor (2)	159
Public methods	159
operator= (1)	159
operator= (2)	159
Inherited public methods	159
Inherited protected methods	160
Chapter 35. IccMessage class	161
IccMessage constructor	161
Constructor	161
Public methods	161
className	161
methodName	161
number	161
summary	162
text	162
Inherited public methods	162
Inherited protected methods	162
Chapter 36. IccPartnerId class	163

IccPartnerId constructors	163
Constructor (1)	163
Constructor (2)	163
Public methods	163
operator= (1)	163
operator= (2)	163
Inherited public methods	163
Inherited protected methods	164
Chapter 37. IccProgram class	165
IccProgram constructors	165
Constructor (1)	165
Constructor (2)	165
Public methods	165
address	165
clearInputMessage	165
entryPoint	166
length	166
link	166
load	166
registerInputMessage	167
setInputMessage	167
unload	167
Inherited public methods	167
Inherited protected methods	168
Enumerations	168
CommitOpt	168
LoadOpt	168
Chapter 38. IccProgramId class	169
IccProgramId constructors	169
Constructor (1)	169
Constructor (2)	169
Public methods	169
operator= (1)	169
operator= (2)	169
Inherited public methods	169
Inherited protected methods	170
Chapter 39. IccRBA class	171
IccRBA constructor	171
Constructor	171
Public methods	171
operator= (1)	171
operator= (2)	171
operator== (1)	171
operator== (2)	171
operator!= (1)	171
operator!= (2)	172
number	172
Inherited public methods	172
Inherited protected methods	172
Chapter 40. IccRecordIndex class	173
IccRecordIndex constructor (protected)	173
Constructor	173

Public methods	173
length	173
type	173
Inherited public methods	173
Inherited protected methods	174
Enumerations	174
Type	174
Chapter 41. IccRequestId class	175
IccRequestId constructors	175
Constructor (1)	175
Constructor (2)	175
Constructor (3)	175
Public methods	175
operator= (1)	175
operator= (2)	175
Inherited public methods	176
Inherited protected methods	176
Chapter 42. IccResource class	177
IccResource constructor (protected)	177
Constructor	177
Public methods	177
actionOnCondition	177
actionOnConditionAsChar	177
actionsOnConditionsText	178
clear	178
condition	178
conditionText	178
get	178
handleEvent	179
id	179
isEDFOn	179
isRouteOptionOn	179
name	179
put	179
routeOption	180
setActionOnAnyCondition	180
setActionOnCondition	180
setActionsOnConditions	180
setEDF	180
setRouteOption (1)	181
setRouteOption (2)	181
Inherited public methods	181
Inherited protected methods	181
Enumerations	182
ActionOnCondition	182
HandleEventReturnOpt	182
ConditionType	182
Chapter 43. IccResourceId class	183
IccResourceId constructors (protected)	183
Constructor (1)	183
Constructor (2)	183
Public methods	183
name	183

nameLength	183
Protected methods	184
operator=	184
Inherited public methods	184
Inherited protected methods	184
Chapter 44. lccRRN class	185
lccRRN constructors	185
Constructor	185
Public methods	185
operator= (1)	185
operator= (2)	185
operator== (1)	185
operator== (2)	185
operator!= (1)	185
operator!= (2)	186
number	186
Inherited public methods	186
Inherited protected methods	186
Chapter 45. lccSemaphore class	187
lccSemaphore constructor	187
Constructor (1)	187
Constructor (2)	187
Public methods	187
lifeTime	187
lock	187
tryLock	188
type	188
unlock.	188
Inherited public methods	188
Inherited protected methods	189
Enumerations	189
LockType	189
LifeTime	189
Chapter 46. lccSession class	191
lccSession constructors (public)	191
Constructor (1)	191
Constructor (2)	191
Constructor (3)	191
lccSession constructor (protected)	191
Constructor	191
Public methods	192
allocate	192
connectProcess (1)	192
connectProcess (2)	192
connectProcess (3)	192
converse.	193
convId	193
errorCode	193
extractProcess	193
flush	194
free	194
get	194
isErrorSet	194

isNoDataSet	194
isSignalSet	194
issueAbend	194
issueConfirmation	195
issueError	195
issuePrepare	195
issueSignal	195
PIPList	195
process	195
put	196
receive	196
send (1)	196
send (2)	196
sendInvite (1)	197
sendInvite (2)	197
sendLast (1)	197
sendLast (2)	197
state	198
stateText	198
syncLevel	198
Inherited public methods	198
Inherited protected methods	199
Enumerations	199
AllocateOpt	199
SendOpt	199
StateOpt	199
SyncLevel	200
Chapter 47. IccStartRequestQ class	201
IccStartRequestQ constructor (protected)	201
Constructor	201
Public methods	201
cancel	201
clearData	201
data	202
instance	202
queueName	202
registerData	202
reset	202
retrieveData	202
returnTermId	203
returnTransId	203
setData	203
setQueueName	203
setReturnTermId (1)	203
setReturnTermId (2)	204
setReturnTransId (1)	204
setReturnTransId (2)	204
setStartOpts	204
start	204
Inherited public methods	205
Inherited protected methods	206
Enumerations	206
RetrieveOpt	206
ProtectOpt	206
CheckOpt	206

Chapter 48. IccSysId class	207
IccSysId constructors	207
Constructor (1)	207
Constructor (2)	207
Public methods	207
operator= (1)	207
operator= (2)	207
Inherited public methods	207
Inherited protected methods	208
Chapter 49. IccSystem class	209
IccSystem constructor (protected)	209
Constructor	209
Public methods	209
appName	209
beginBrowse (1)	209
beginBrowse (2)	209
dateFormat	210
endBrowse	210
freeStorage	210
getFile (1)	210
getFile (2)	210
getNextFile	211
getStorage	211
instance	211
operatingSystem	211
operatingSystemLevel	212
release	212
releaseText	212
sysId	212
workArea	212
Inherited public methods	213
Inherited protected methods	213
Enumerations	213
ResourceType	213
Chapter 50. IccTask class	215
IccTask Constructor (protected)	215
Constructor	215
Public methods	215
abend	215
abendData	215
commitUOW	215
delay	216
dump	216
enterTrace	217
facilityType	217
freeStorage	217
getStorage	217
instance	218
isCommandSecurityOn	218
isCommitSupported	218
isResourceSecurityOn	218
isRestarted	218
isStartDataAvailable	219
number	219

principalSysId	219
priority	219
rollBackUOW	219
setDumpOpts	219
setPriority	220
setWaitText	220
startType	220
suspend	220
transId	220
triggerDataQueueId	220
userId	221
waitExternal	221
waitOnAlarm	221
workArea	222
Inherited public methods	222
Inherited protected methods	222
Enumerations	222
AbendHandlerOpt	222
AbendDumpOpt	223
DumpOpts	223
FacilityType	223
StartType	223
StorageOpts	224
TraceOpt	224
WaitPostType	224
WaitPurgeability	224
Chapter 51. IccTempStore class	225
IccTempStore constructors	225
Constructor (1)	225
Constructor (2)	225
Public methods	225
clear	225
empty	226
get	226
numberOfItems	226
put	226
readItem	226
readNextItem	226
rewriteItem	227
writeItem (1)	227
writeItem (2)	227
Inherited public methods	228
Inherited protected methods	228
Enumerations	228
Location	228
NoSpaceOpt	228
Chapter 52. IccTempStoreId class	229
IccTempStoreId constructors	229
Constructor (1)	229
Constructor (2)	229
Public methods	229
operator= (1)	229
operator= (2)	229
Inherited public methods	229

Inherited protected methods	230
Chapter 53. lccTermId class	231
lccTermId constructors	231
Constructor (1)	231
Constructor (2)	231
Public methods	231
operator= (1)	231
operator= (2)	231
Inherited public methods	231
Inherited protected methods	232
Chapter 54. lccTerminal class	233
lccTerminal constructor (protected)	233
Constructor	233
Public methods	233
AID	233
clear	233
cursor	233
data	233
erase	234
freeKeyboard	234
get	234
height	234
inputCursor	234
instance	234
line	235
netName	235
operator<< (1)	235
operator<< (2)	235
operator<< (3)	235
operator<< (4)	235
operator<< (5)	235
operator<< (6)	235
operator<< (7)	236
operator<< (8)	236
operator<< (9)	236
operator<< (10)	236
operator<< (11)	236
operator<< (12)	236
operator<< (13)	236
operator<< (14)	236
operator<< (15)	236
operator<< (16)	237
operator<< (17)	237
operator<< (18)	237
put	237
receive	237
receive3270Data	237
send (1)	238
send (2)	238
send (3)	238
send (4)	238
send3270Data (1)	239
send3270Data (2)	239
send3270Data (3)	239

send3270Data (4)	239
sendLine (1)	239
sendLine (2)	240
sendLine (3)	240
sendLine (4)	240
setColor	240
setCursor (1)	240
setCursor (2)	241
setHighlight	241
setLine	241
setNewLine	241
setNextCommArea	241
setNextInputMessage	242
setNextTransId	242
signoff	242
signon (1)	242
signon (2)	242
waitForAID (1)	243
waitForAID (2)	243
width	243
workArea	243
Inherited public methods	244
Inherited protected methods	244
Enumerations	244
AIDVal	244
Case	244
Color	244
Highlight	245
NextTransIdOpt	245
Chapter 55. IccTerminalData class	247
IccTerminalData constructor (protected)	247
Constructor	247
Public methods	247
alternateHeight	247
alternateWidth	247
defaultHeight	248
defaultWidth	248
graphicCharCodeSet	248
graphicCharSetId	248
isAPLKeyboard	248
isAPLText	248
isBTrans	249
isColor	249
isEWA	249
isExtended3270	249
isFieldOutline	249
isGoodMorning	250
isHighlight	250
isKatakana	250
isMSRControl	250
isPS	250
isSOSI	250
isTextKeyboard	251
isTextPrint	251
isValidation	251

Inherited public methods	251
Inherited protected methods	252
Chapter 56. lccTime class	253
lccTime constructor (protected)	253
Constructor	253
Public methods	253
hours	253
minutes	253
seconds	253
timInHours	253
timInMinutes	254
timInSeconds	254
type	254
Inherited public methods	254
Inherited protected methods	254
Enumerations	255
Type	255
Chapter 57. lccTimeInterval class	257
lccTimeInterval constructors	257
Constructor (1)	257
Constructor (2)	257
Public methods	257
operator=	257
set	257
Inherited public methods	258
Inherited protected methods	258
Chapter 58. lccTimeOfDay class	259
lccTimeOfDay constructors	259
Constructor (1)	259
Constructor (2)	259
Public methods	259
operator=	259
set	259
Inherited public methods	260
Inherited protected methods	260
Chapter 59. lccTPNameId class	261
lccTPNameId constructors	261
Constructor (1)	261
Constructor (2)	261
Public methods	261
operator= (1)	261
operator= (2)	261
Inherited public methods	261
Inherited protected methods	262
Chapter 60. lccTransId class	263
lccTransId constructors	263
Constructor (1)	263
Constructor (2)	263
Public methods	263
operator= (1)	263
operator= (2)	263

Inherited public methods	263
Inherited protected methods	264
Chapter 61. IccUser class	265
IccUser constructors	265
Constructor (1)	265
Constructor (2)	265
Public methods	265
changePassword.	265
daysUntilPasswordExpires	266
ESMReason	266
ESMResponse	266
groupId	266
invalidPasswordAttempts	266
language	266
lastPasswordChange	266
lastUseTime	266
passwordExpiration	267
setLanguage	267
verifyPassword	267
Inherited public methods	267
Inherited protected methods	268
Chapter 62. IccUserId class	269
IccUserId constructors.	269
Constructor (1)	269
Constructor (2)	269
Public methods	269
operator= (1)	269
operator= (2)	269
Inherited public methods	269
Inherited protected methods	270
Chapter 63. IccValue structure	271
Enumeration	271
CVDA.	271
Chapter 64. main function.	275

Part 4. Appendixes 277

Appendix A. Mapping EXEC CICS calls to Foundation Class methods	279
Appendix B. Mapping Foundation Class methods to EXEC CICS calls	285
Appendix C. Output from sample programs	291
ICC\$BUF (IBUF).	291
ICC\$CLK (ICLK).	291
ICC\$DAT (IDAT).	291
ICC\$EXC1 (IEX1)	292
ICC\$EXC2 (IEX2)	292
ICC\$EXC3 (IEX3)	292
ICC\$FIL (IFIL).	293
ICC\$HEL (IHEL).	293
ICC\$JRN (IJRN).	293
ICC\$PRG1 (IPR1)	294

First Screen	294
Second Screen	294
ICC\$RES1 (IRS1)	294
ICC\$RES2 (IRS2)	295
ICC\$SEM (ISEM)	295
ICC\$SES1 (ISE1)	295
ICC\$SES2 (ISE2)	296
ICC\$SRQ1 (ISR1)	296
ICC\$SRQ2 (ISR2)	296
ICC\$SYS (ISYS)	297
ICC\$TMP (ITMP)	297
ICC\$TRM (ITRM)	298
ICC\$TSK (ITSK)	298
Glossary	299
Bibliography	301
The CICS Transaction Server for z/OS library	301
The entitlement set	301
PDF-only books	301
Other CICS books	303
Related books.	303
C++ Programming	303
CICS client manuals	303
Determining if a publication is current	304
Accessibility	305
Index	307
Notices	345
Trademarks.	346
Sending your comments to IBM	347

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,” on page 1 describes how to install the product and check that the installation is complete.
- Part 2, Using the CICS foundation classes,” on page 13 describes the classes and how to use them.
- Part 3, Foundation Classes—reference,” on page 65 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,” on page 279 maps EXEC CICS calls to the foundation class methods detailed in this book...
- ... and Appendix B, Mapping Foundation Class methods to EXEC CICS calls,” on page 285 maps them the other way — foundation class methods to EXEC CICS calls.
- Appendix C, Output from sample programs,” on page 291 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 3 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 299 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 301, 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.

C++ Header File	Classes Defined in this Header
ICCABDEH	IccAbendData
ICCBASEH	IccBase
ICCBUFEH	IccBuf
ICCCLKEH	IccClock
ICCCNDEH	IccCondition (struct)
ICCCONEH	IccConsole
ICCCTLEH	IccControl
ICCDATEH	IccDataQueue
ICCEH	see 1
ICCEVTEH	IccEvent
ICCEXCEH	IccException
ICCFILEH	IccFile
ICCFLIEH	IccFileIterator
ICGGLBEH	Icc (struct) (global functions)
ICJRNEH	IccJournal
ICMSGEH	IccMessage
ICCPRGEH	IccProgram
ICCRECEH	IccRecordIndex, IccKey, IccRBA and IccRRN
ICCRESEH	IccResource
ICCRIDEH	IccResourceId + subclasses (such as IccConvId)
ICCSEMEH	IccSemaphore
ICCSESEH	IccSession
ICCSRQEH	IccStartRequestQ
ICCSYSEH	IccSystem
ICCTIMEH	IccTime, IccAbsTime, IccTimeInterval, IccTimeOfDay
ICCTMDEH	IccTerminalData
ICCTMPEH	IccTempStore
ICCTRMEH	IccTerminal
ICCTSKEH	IccTask
ICCUSREH	IccUser
ICCVALEH	IccValue (struct)

Notes:

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

Installed contents

-
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: CICSTS31.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: CICSTS31.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: CICSTS31.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 CICSTS31.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 CICSTS31.CICS.SDFHPROC.

ICCFCCL contains the Job Control Language needed to compile, pre-link and link a CICS user application. Before using ICCFCCL you may find it necessary to perform some customization to conform to your installation standards. See also "Compiling Programs" on page 49.

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

CICSTS31.CICS.SDFHSDCK contains the member
ICCFCIMP - 'sidedeck' containing import control statements

CICSTS31.CICS.SDFHPROC contains the members
ICCFCC - JCL to compile a CFC user program
ICCFCCL - JCL to compile, prelink and link a CFC user program
ICCFCGL - JCL to compile and link an XPLINK program that uses CFC libraries.

#

ICCFCL - JCL to prelink and link a CFC user program

CICSTS31.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 17 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).

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

The first line includes the header file, ICCEH, which includes the header files for all the CICS Foundation Class definitions. Note that it is coded as `icceh.hpp` to preserve cross-platform, C++ language conventions.

The second line includes the supplied program stub. This stub contains the **main** function, which is the point of entry for any program that uses the supplied classes and is responsible for initializing them correctly. (See Chapter 64, “main function,” on page 275 for more details). You are strongly advised to use the stub provided but you may in certain cases tailor this stub to your own requirements. The stub initializes the class environment, creates the program control object, then invokes the **run** method, which is where the application program should live.

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

The code that controls the program flow resides not in the **main** function but in the **run** method of a class derived from **IccControl** (see Chapter 20, “IccControl class,” on page 111). The user can define their own subclass of **IccControl** or, as here, use the default one – **IccUserControl**, which is defined in ICCMAIN – and just provide a definition for the **run** method.

```
    IccTerminal* pTerm = terminal();
```

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

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

The **erase** method clears the current contents of the terminal.

Hello World

```
pTerm->send(10, 35, 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.

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

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

```
    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 49 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.

Part 2. Using the CICS foundation classes

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 65.

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:

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

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

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

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

Here we deal with pointers to objects instead of the objects themselves. The lifetime of the object outlives the scope in which it was created. In the 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:

```
{
    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 61.

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:

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

Alternatively, you can do this using dynamic object creation:

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

Deleting an object

When an object is destroyed its destructor function, which has the same name as the class preceded with ~(tilde), is 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.

Chapter 5. Overview of the foundation classes

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 18
- Resource classes” on page 19
- Support Classes” on page 20.

See Part 3, “Foundation Classes—reference,” on page 65 for more detailed information on the Foundation Classes.

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

Base classes

```
iccBase
  iccRecordIndex
  iccResource
    iccControl
    iccTime
  iccResourceId
```

Figure 1. Base classes

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

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 19.

iccResourceId

The base class for all table entry (resource name) classes, such as **iccFileId** and **iccTempStoreId**.

Base classes

IccTime

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

Resource identification classes

IccBase

- IccResourceId**
- IccConvId**
- IccDataQueueId**
- IccFileId**
- IccGroupId**
- IccJournalId**
- IccJournalTypeId**
- IccLockId**
- IccPartnerId**
- IccProgramId**
- IccRequestId**
 - IccAlarmRequestId**
- IccSysId**
- IccTempStoreId**
- IccTermId**
- IccTPNameId**
- IccTransId**
- IccUserId**

Figure 2. 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 19, contain resource identification classes. For example, an **IccFile** object contains an **IccFileId** object. You must use the resource object, not the resource identification object to operate on a CICS resource. For example, you must use **IccFile**, rather than **IccFileId** to read a record from a file.

Class	CICS resource	CICS table
IccAlarmRequestId	alarm request	
IccConvId	conversation	

Class	CICS resource	CICS table
IccDataQueueId	data queue	
IccFileId	file	FCT
IccGroupId	group	
IccJournalId	journal	
IccJournalTypeId	journal type	
IccLockId	(Not applicable)	
IccPartnerId	APPC partner definition files	
IccProgramId	program	PPT
IccRequestId	request	
IccSysId	remote system	
IccTempStoreId	temporary storage	TST
IccTermId	terminal	TCT
IccTPNameId	remote APPC TP name	
IccTransId	transaction	PCT
IccUserId	user	

Resource classes

```

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

```

Figure 3. Resource classes

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

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

All CICS resource classes inherit from the **IccResource** base class. For example, any operation on a CICS resource may raise a CICS condition; the **condition** method of **IccResource** (see page 178) can interrogate it.

Resource classes

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

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

Support Classes

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

Figure 4. Support classes

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

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

Resource class	Description
IccTimeOfDay	Time of day (for example, five minutes past six)

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

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

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

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

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

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

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

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

The **IccBuf** class allows easy manipulation of buffers, such as file record buffers, transient data record buffers, and COMMAREAs (for more information on **IccBuf** class see Chapter 6, Buffer objects," on page 25).

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.

Using CICS resources

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

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

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

as:

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

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

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

Singleton classes

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

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

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

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

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

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

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

Calling methods on a resource object

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

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

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

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 objects 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.

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

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

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

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

The **IccTempStore** object calls its **writelnItem** 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 programs **IccBuf** object:

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

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

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

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

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

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

You can find further examples of this syntax in the samples presented in the following 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 34
- Starting transactions asynchronously on page 36
- Transient Data on page 39
- Temporary storage on page 41
- Terminal control on page 43
- Time and date services on page 45

File control

The file control classes – **IccFile**, **IccFileid**, **IccKey**, **IccRBA**, and **IccRRN** – allow you to read, write, update and delete records in files. In addition, **IccFileIterator** class allows you to browse through all the records in a file.

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

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

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

KSDS

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

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

When browsing, records are found in their logical order.

ESDS

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

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

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

RRDS file

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

Reading records

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

The **readRecord** method of **IccFile** class 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 31 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 **IccKey** object to access records. A complete key is a character string that uniquely identifies a record. Every record can be separately identified by its complete key.

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

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

Writing ESDS records

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

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

Writing RRDS records

Use the **writeRecord** method to add a new ESDS record. After writing the record you can use the **number** method on the **IccRBA** object to discover the assigned relative byte address for the record you have just written.

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

Updating records

Updating a record is also known as rewriting a record. Before updating a record you must first read it, using **readRecord** method in update mode. This locks the record so that nobody else can change it.

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

For an example, see code fragment: "Read record for update" on page 34.

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

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

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

Deleting records

Records can never be deleted from an ESDS file.

Deleting normal records

The **deleteRecord** method of **IccFile** class deletes one or more records, provided they are not locked by virtue of being in update mode. The records to be deleted are defined by the **IccKey** or **IccRRN** object.

File control

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 ascending order of key~ on page 33.

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.

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

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

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

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

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

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

```

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

```

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

```

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

```

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

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

```

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

```

The loop shown here lists to the terminal, using **sendLine**, all the records in ascending order of key. It uses an **IccFileIterator** object to browse the records. It starts by setting the minimum value for the key which, as it happens, does not 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.

```

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.

File control

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

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

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

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

```
return;
}
```

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

See Appendix C, "Output from sample programs," on page 291 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 165 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.

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

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

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

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

created to be used as a communication area between programs.

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

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

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

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

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

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

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

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

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

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

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

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

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

Program control

```
    return;  
};
```

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

See Appendix C, “Output from sample programs,” on page 291 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

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

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).

Starting transactions asynchronously

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:

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

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

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

Here we are creating a number of objects:

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

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

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

remoteTermId

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

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

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

buffer

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

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

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

This code fragment prepares the start data that is passed when we issue a start request. The **setRouteOption** says we will issue the start request on the remote system, ICC2. The **registerData** method associates an **IccBuf** object that will contain the start data (the contents of the **IccBuf** object are not extracted until we 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.

```
buffer = startQ->data();
term->sendLine( Start buffer contents = [%s], buffer.dataArea() );
term->sendLine( Start queue= [%s], startQ->queueName() );
term->sendLine( Start rtn = [%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 291 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 41.

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.

Transient Data

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.

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

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

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

This defines some buffer for the sample program.

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

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

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

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

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

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

```

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

```

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

```

    return;
}

```

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

See Appendix C, "Output from sample programs," on page 291 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 (**writeltem** method)
- Update an item in the temporary storage queue (**rewriteltem** 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 39.

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 **rewriteltem** method. See "Updating items" on page 42 for more information.

The **writeltem** 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 **rewriteltem** 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.

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

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

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

This defines some buffer for the sample program.

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

return;
}

```

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

See Appendix C, "Output from sample programs," on page 291 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** on page 247 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.

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

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

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

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

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

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

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

This sends a blank line to the screen.

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

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

```
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.

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

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

```
    return;
}
```

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

See Appendix C, "Output from sample programs," on page 291 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 `ICC$CLK`. The sample is presented here without the terminal IO requests.

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

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

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 291 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:

```
//ICC$HEL JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//PROCLIB JCLLIB ORDER=(CICSTS31.CICS.SDFHPROC)
//ICC$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 53
- Platform differences” on page 56

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*, GC34-6442.

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):

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

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

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

Conditions, errors, exceptions

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

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

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

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

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

The output from this CICS program is as follows:

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

The CICS C++ Foundation Classes do not throw **char*** exceptions as in the 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, if an **IccBuf** object with too much data is passed to the **writeln** 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.

```
#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 291 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 65.

Conditions, errors, exceptions

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.

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:

```
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:

```
IccTempStore temp(TEMP1234);
IccBuf buf(40);
temp.setActionOnCondition(IccResource::noAction,
                        IccCondition::QIDERR);

buf = temp.readNextItem();
switch (temp.condition())
{
case IccCondition::QIDERR:
    //do whatever here
    :
default:
    //do something else here
}
```

Automatic condition handling (callHandleEvent)

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

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

When a call to any method on object temp causes CICS to raise the QIDERR condition, **handleEvent** method is automatically called. As the **handleEvent** method

is only a virtual method, this call is only useful if the object belongs to a subclass of **IccTempStore** and the **handleEvent** method has been overridden.

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

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

Now implement the **handleEvent** method.

```
IccResource::HandleEventReturnOpt MyTempStore::handleEvent(IccEvent& event)
{
    switch (event.condition())
    {
        case ...

        :
        case IccCondition::QIDERR:
            //Handle QIDERR condition here.

        :
            //
        default:
            return rAbendTask;
    }
}
```

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

Exception handling (throwException)

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

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

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

```
try
{
    buf = temp.readNextItem();

    :
}
catch (IccException& exception)
{
    //Exception handling code

    :
}
```

Exception handling

An exception is thrown if any of the methods inside the try block raise the QIDERR condition for object temp. When an exception is thrown, C++ unwinds the stack and resumes execution at an appropriate **catch** block – it is not possible to resume within the **try** block. For a fuller example 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 51 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 65. 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

journaling 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:

```
IccJournal journal17(7); //No good on CICS OS/2
```

or

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

If you initialize your application with `fsEnforce` selected (see `initializeEnvironment` on page 67) 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 65 fully documents all platform restrictions.

Method level

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

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

Here method **programId** executes correctly on CICS 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.

Platform differences

```
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

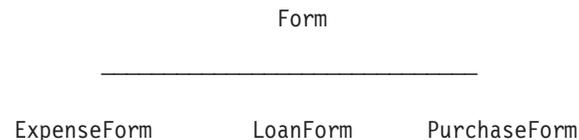
This chapter describes the following:

- Polymorphic Behavior
- Storage management on page 61
- Parameter passing conventions on page 62
- Scope of data in lccBuf reference returned from read methods on page 63

Polymorphic Behavior

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

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



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

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

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

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

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

Miscellaneous

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

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

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

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

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

These virtual methods are **not** supported by any subclasses of **IccResource** except those in the table 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 ICC\$RES2. It is presented here without the terminal IO requests.

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

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

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

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

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

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

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

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

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

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

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

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

Storage management

C++ objects are usually stored on the stack or heap— see “Creating an object” on page 15. 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).

Miscellaneous

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

This facility is affected by the **Icc::initializeEnvironment** call that must be issued before using the Foundation Classes. (This function is called from the default **main** function—see Chapter 64, *main function*,” on page 275.)

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

cmmDefault

The default action is platform dependent:

MVS/ESA

same as **cmmNonCICS** - see below.

UNIX same as **cmmCICS** - see below.

OS/2 same as **cmmCICS** - see below.

cmmNonCICS

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

cmmCICS

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

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

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

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

Parameter passing conventions

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

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

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

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

Using the 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:

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

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

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

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

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

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

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

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

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

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

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

Miscellaneous

Part 3. Foundation Classes—reference

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 **lcc**. You are advised not to create your own classes with this prefix.

lcc structure contains some functions and enumerations that are widely applicable. **lccValue** 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 **lccBase** 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. lcc... 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 **lcc**.
5. Compound names have no separators, but have capital letters to demark the beginning of second and subsequent words, as in **lccJournalTyped**.
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 13.

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

```
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

```
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 275. All OO CICS programs should use this stub or a close equivalent.

conditionText

```
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

lcc

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

mem

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

fam

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

EDF

A boolean that indicates whether EDF tracing is initially on.

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 275. All OO CICS programs should use this stub or a close equivalent.

isClassMemoryMgmtOn

```
static Bool isClassMemoryMgmtOn()
```

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

isEDFOn

```
static Bool isEDFOn()
```

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 177 and **Execution Diagnostic Facility** on page 50).

isFamilySubsetEnforcementOn

```
static Bool isFamilySubsetEnforcementOn()
```

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

returnToCICS

```
static void returnToCICS()
```

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 275. All OO CICS programs should use this stub or a close equivalent.

setEDF

```
static void setEDF(icc::Bool onOff = off)
```

onOff

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

Sets EDF tracing on or off at the global level.

unknownException

```
static void unknownException()
```

This function is called by the **main** function in ICCMAIN header file (see Chapter 64, *main function*,” on page 275) 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:

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

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

BoolSet

```
trueFalse
yesNo
onOff
```

ClassMemoryMgmt

```
cmmDefault
```

The defaults for the different platforms are:

MVS/ESA

```
cmmNonCICS
```

OS/2 cmmCICS

UNIX cmmCICS

lcc

cmmNonCICS

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

In MVS/ESA 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 **lccBase** 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 pageGetOpt” on page 70.

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

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

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

SUBSPACE

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

BASESPACE

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

NOTAPPLIC

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

Conditions

INVREQ

ASRAStorageType

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

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

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

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

USER

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

READONLY

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

NOTAPPLIC

One of:

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

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

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

Method

setActionOnCondition
setActionsOnConditions
setEDF

Class

IccResource
IccResource
IccResource

Inherited protected methods**Method**

setClassName
setCustomClassNum

Class

IccBase
IccBase

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 milliseconds 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

timelnHours

```
unsigned long timelnHours()
```

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

timelnMinutes

```
unsigned long timelnMinutes()
```

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

timelnSeconds

```
unsigned long timelnSeconds()
```

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

year

IccAbsTime

unsigned long year()

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

Conditions

INVREQ

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

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

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

Inherited protected methods

Method	Class
operator=	IccResourceId
setClassName	IccBase

Method

setCustomClassNum

Class

IccBase

IccAlarmRequestId

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 `inabendCode` on page 71.

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 87.

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.

lccBase

customClassNum

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 67).

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 **lccTempStore**, 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:

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

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

NameOpt

See `className` on page 85.

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*.

Chapter 16. `IccBuf` class

`IccBase`
`IccBuf`

`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 25.

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 25.

Constructor (3)

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

text

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

type

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

lccBuf

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

Constructor (4)

lccBuf(const lccBuf& buffer)

buffer

A reference to an **lccBuf** object that is to be copied into the new object. The copy constructor—creates a new **lccBuf** object that is a copy of the given object. The created **lccBuf** object **always** has an internal data area.

Public methods

append (1)

**lccBuf& 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.

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

append (2)

**lccBuf& 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)

**lccBuf& 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 97.

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 97.

lccBuf

dataLength

unsigned long dataLength() const

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

insert

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

length

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

dataArea

The start of the source data to be inserted into the **lccBuf** 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 **lccBuf** object.

isFMHContained

lcc::Bool isFMHContained() const

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

operator const char*

operator const char*() const

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

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

operator= (1)

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

buffer

A reference to an **lccBuf** object.

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

operator= (2)

lccBuf& 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.

lccBuf

operator<< (2)

operator<<(const char* *text*)

Appends a string.

operator<< (3)

operator<<(char *ch*)

Appends a character.

operator<< (4)

operator<<(signed char *ch*)

Appends a character.

operator<< (5)

operator<<(unsigned char *ch*)

Appends a character.

operator<< (6)

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

Appends a string.

operator<< (7)

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

Appends a string.

operator<< (8)

operator<<(short *num*)

Appends a short.

operator<< (9)

operator<<(unsigned short *num*)

Appends an unsigned short.

operator<< (10)

operator<<(long *num*)

Appends a long.

operator<< (11)

operator<<(unsigned long *num*)

Appends an unsigned long.

operator<< (12)

operator<<(int *num*)

Appends an integer.

operator<< (13)

operator<<(float *num*)

Appends a float.

operator<< (14)

operator<<(double *num*)

Appends a double.

operator<< (15)

operator<<(long double *num*)

Appends a long double.

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 25.

replace

lccBuf

lccBuf& 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 **lccBuf** 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 **lccBuf** 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 **lccBuf** 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(**lcc::Bool** *yesNo* = lcc::yes)

yesNo

A boolean, defined in **lcc** 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

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

Inherited protected methods

Method	Class
setClassName	lccBase
setCustomClassNum	lccBase

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** 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')

lccClock

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

```
const IccAlarmRequestId& setAlarm (const IccTime& time,
                                   const IccRequestId* reqId = 0)
```

time

A reference to an **IccTime** object that contains time information. As **IccTime** is an abstract class *time* is, in practise, an object of class **IccAbsTime**, **IccTimeOfDay**, or **IccTimeInterval**.

reqId

An optional pointer to an **IccRequestId** object that is used to identify this particular alarm request.

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 221 of class **IccTask**.

Conditions

EXPIRED, INVREQ

time

```
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

```
void update()
```

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

year

```
unsigned long year()
```

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

Conditions

INVREQ

Inherited public methods

Method

actionOnCondition
 actionOnConditionAsChar
 actionsOnConditionsText
 classType

Class

IccResource
 IccResource
 IccResource
 IccBase

IccClock

Method	Class
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

DateFormat

defaultFormat
DDMMYY
MMDDYY
YYDDD
YYDDMM
YYMMDD
DDMMYYYY
MMDDYYYY
YYYYDDD
YYYYDDMM
YYYYMMDD

DayOfWeek

Indicates the day of the week.

Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday

MonthOfYear

Indicates the month of the year.

January

February
March
April
May
June
July
August
September
October
November
December

UpdateMode

Indicates whether the clock is automatically updated.

manual

The clock initially holds the time at which it was created. It is subsequently updated only when an **update** method call is made.

automatic

The clock is updated to the current CICS time and date whenever any time or date method is called (for example, **daysSince1900**).

Chapter 18. IccCondition structure

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

Header file: ICCCNDEH

Enumerations

Codes

The possible values are:

	Value		Value		Value
0	NORMAL	35	TSIOERR	70	NOTAUTH
1	ERROR	36	MAPFAIL	—	
2	RDATT	37	INVERRTERM	72	SUPPRESSED
3	WRBRK	38	INVMPSZ	—	
4	ICCEOF	39	IGREQID	—	
5	EODS	40	OVERFLOW	75	RESIDERR
6	EOC	41	INVLDC	—	
7	INBFMH	42	NOSTG	—	
8	ENDINPT	43	JIDERR	—	
9	NONVAL	44	QIDERR	—	
10	NOSTART	45	NOJBUFSP	80	NOSPOOL
11	TERMIDERR	46	DSSTAT	81	TERMERR
12	FILENOTFOUND	47	SELNERR	82	ROLLEDBACK
13	NOTFND	48	FUNCERR	83	END
14	DUPREC	49	UNEXPIN	84	DISABLED
15	DUPKEY	50	NOPASSBKRD	85	ALLOCERR
16	INVREQ	51	NOPASSBKWR	86	STRELERR
17	IOERR	—		87	OPENERR
18	NOSPACE	53	SYSIDERR	88	SPOLBUSY
19	NOTOPEN	54	ISCINVREQ	89	SPOLERR
20	ENDFILE	55	ENQBUSY	90	NODEIDERR
21	ILLOGIC	56	ENVDEFERR	91	TASKIDERR
22	LENGERR	57	IGREQCD	92	TCIDERR
23	QZERO	58	SESSIONERR	93	DSNNOTFOUND
24	SIGNAL	59	SYSBUSY	94	LOADING
25	QBUSY	60	SESSBUSY	95	MODELIDERR
26	ITEMERR	61	NOTALLOC	96	OUTDESCERR
27	PGMIDERR	62	CBIDERR	97	PARTNERIDERR
28	TRANSIDERR	63	INVEXITREQ	98	PROFILEIDERR
29	ENDDATA	64	INVPARTNSET	99	NETNAMEIDERR
30	INVTSREQ	65	INVPARTN	100	LOCKED
31	EXPIRED	66	PARTNFAIL	101	RECORDBUSY
32	RETPAGE	—		102	UOWNOTFOUND
33	RTEFAIL	—		103	UOWLNOTFOUND
34	RTESOME	69	USERIDERR		

Range

maxValue

The highest CICS condition, currently 103.

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 71.

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 59.

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.

Conditions

INVREQ, LENGERR, EXPIRED

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.

Conditions

INVREQ, LENGERR, EXPIRED

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

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.

lccControl

Conditions

INVREQ

console

lccConsole* console()

Returns a pointer to the single **lccConsole** object. If this object has not yet been created, this method creates the object before returning a pointer to it.

initData

const lccBuf& initData()

Returns a reference to an **lccBuf** object that contains the initialization parameters specified for the program in the INITPARM system initialization parameter.

Conditions

INVREQ

instance

static lccControl* instance()

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

isCreated

static lcc::Bool isCreated()

Returns a boolean value that indicates whether the **lccControl** object already exists. Possible values are true or false.

programId

const lccProgramId& programId()

Returns a reference to an **lccProgramId** 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 111).

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

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

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

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

Inherited protected methods

Method

operator=
setClassName
setCustomClassNum

Class

IccResourceId
IccBase
IccBase

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 59.

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()

IccDataQueue

A synonym for **readItem**. See “Polymorphic Behavior” on page 59.

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 **writeltem**. See “Polymorphic Behavior” on page 59.

readItem

const IccBuf& readItem()

Returns a reference to an **IccBuf** object that contains one item read from the data queue.

Conditions

IOERR, ISCVREQ, LENGERR, NOTAUTH, NOTOPEN, QBUSY, QIDERR, QZERO, SYSIDERR, DISABLED, INVREQ

writeltem (1)

void writeltem(const IccBuf& item)

item

A reference to an **IccBuf** object that contains data to be written to the queue.

writeltem (2)

void writeltem(const char* text)

text

Text that is to be written to the queue.

Writes an item of data to the queue.

Conditions

IOERR, ISCVREQ, LENGERR, NOSPACE, NOTAUTH, NOTOPEN, QIDERR, SYSIDERR, DISABLED, INVREQ

Inherited public methods

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource

Method

isEDFOn
isRouteOptionOn
name
operator delete
operator new
routeOption
setActionOnAnyCondition
setActionOnCondition
setActionsOnConditions
setEDF
setRouteOption

Class

IccResource
IccResource
IccResource
IccBase
IccBase
IccResource
IccResource
IccResource
IccResource
IccResource
IccResource

Inherited protected methods**Method**

setClassName
setCustomClassNum

Class

IccBase
IccBase

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

Method

classType
className
customClassNum
name
nameLength
operator delete
operator new

Class

IccBase
IccBase
IccBase
IccResourceId
IccResourceId
IccBase
IccBase

Inherited protected methods

Method

operator=
setClassName
setCustomClassNum

Class

IccResourceId
IccBase
IccBase

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 85 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

const char* conditionText() const

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

methodName

const char* methodName() const

Returns the name of the method responsible for this event.

summary

const char* summary()

Returns a summary of the CICS event in the form:

CICS event summary: IccDataQueue::readItem condition=23 (QZERO) minor=0

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

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

Method	Class
<code>className</code>	<code>IccBase</code>
<code>classType</code>	<code>IccBase</code>
<code>customClassNum</code>	<code>IccBase</code>
<code>operator delete</code>	<code>IccBase</code>
<code>operator new</code>	<code>IccBase</code>

Inherited protected methods

Method	Class
<code>setClassName</code>	<code>IccBase</code>
<code>setCustomClassNum</code>	<code>IccBase</code>

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 `writeltem` 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.

IccException

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.
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 56 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)

IccFile (**const IccFileId&** *id*,
IccRecordIndex* *index* = 0)

id

A reference to the **IccFileId** object that identifies which file is being operated on

index

An optional pointer to the **IccRecordIndex** object that identifies which record in the file is being operated on.

Constructor (2)

IccFile (**const char*** *fileName*,
IccRecordIndex* *index* = 0)

fileName

The 8-character name of the file

index

An optional pointer to the **IccRecordIndex** object that identifies which record in the file is being operated on.

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

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method in *abendCode* on page 71.

access

unsigned long access(Icc::GetOpt opt =Icc::object)

lccFile

opt

An enumeration, defined in **lcc** 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.

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

accessMethod

lccValue::CVDA accessMethod(lcc::GetOpt opt = lcc::object)

opt

See **access** method.

Returns an enumeration, defined in **lccValue**, 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 **lccRecordIndex** 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)

void endInsert()

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

isAddable

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

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

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

opt

See **access** method.

Indicates whether the records in the file can be deleted.

lccFile

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isEmptyOnOpen

lcc::Bool isEmptyOnOpen(lcc::GetOpt opt = lcc::object)

opt

See **access** method.

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

lcc::Bool isReadable(lcc::GetOpt opt = lcc::object)

opt

See **access** method.

Indicates whether the file records can be read.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isRecoverable

lcc::Bool isRecoverable(lcc::GetOpt opt = lcc::object)

opt

See **access** method.

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

isUpdatable

lcc::Bool isUpdatable(lcc::GetOpt opt = lcc::object)

opt

See **access** method.

Indicates whether the file can be updated.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

keyLength

unsigned long keyLength(lcc::GetOpt opt = lcc::object)

opt

See **access** method.

Returns the length of the search key.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

keyPosition

long keyPosition(Icc::GetOpt *opt* = Icc::object)

opt

See **access** method.

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

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

lccFile

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

Conditions

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

recordFormat

lccValue::CVDA recordFormat(lcc::GetOpt opt = lcc::object)

opt

See **access** method.

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

FIXED

The records are of fixed length.

UNDEFINED (BDAM data sets only)

The format of records on the file is undefined.

VARIABLE

The records are of variable length. If the file is associated with a data table, the record format is always variable length, even if the source data set contains fixed-length records.

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

recordIndex

lccRecordIndex* recordIndex() const

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

recordLength

unsigned long recordLength(lcc::GetOpt opt = lcc::object)

opt

See **access** method.

Returns the length of the current record.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

registerRecordIndex

void registerRecordIndex(lccRecordIndex* index)

index

A pointer to an **lccKey**, **lccRBA**, or **lccRRN** object that will be used by methods such as **readRecord**, **writeRecord**, etc..

rewriteRecord

```
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

```
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:

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

Conditions

FILENOTFOUND, INVREQ, IOERR, NOTAUTH

setEmptyOnOpen

```
void setEmptyOnOpen(Icc::Bool trueFalse)
```

Specifies whether or not to make the file empty when it is next opened.

Conditions

FILENOTFOUND, INVREQ, IOERR, NOTAUTH

setStatus

```
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

IccFile

type

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.
VRDS	The data set is a variable relative record data set.

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

unlockRecord

void unlockRecord(unsigned long updateToken = 0)

updateToken

A token that indicates which previous **readRecord** update request is to be unlocked.

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

Conditions

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

writeRecord

void writeRecord(const IccBuf& buffer)

buffer

A reference to the **IccBuf** object that holds the data that is to be written into the record.

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

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase

Method	Class
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

Access

readable	File records can be read by CICS tasks.
notReadable	File records cannot be read by CICS tasks.
browsable	File records can be browsed by CICS tasks.
notBrowsable	File records cannot be browsed by CICS tasks.
addable	Records can be added to the file by CICS tasks.
notAddable	Records cannot be added to the file by CICS tasks.
updatable	Records in the file can be updated by CICS tasks.
notUpdatable	Records in the file cannot be updated by CICS tasks.
deletable	Records in the file can be deleted by CICS tasks.
notDeletable	Records in the file cannot be deleted by CICS tasks.
fullAccess	Equivalent to readable AND browsable AND addable AND updatable AND deletable.
noAccess	Equivalent to notReadable AND notBrowsable AND notAddable AND notUpdatable AND notDeletable.

ReadMode

The mode in which a file is read.

normal	No update is to be performed (that is, read-only mode)
update	The record is to be updated. The record is locked by CICS until: <ul style="list-style-type: none"> it is rewritten using the rewriteRecord method <i>or</i>

IccFile

- 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
gteqToKey

The search only finds an exact match.
The search finds either an exact match or the next record in search order.

Status

open

File is open, ready for read/write requests by CICS tasks.

closed

File is closed, and is therefore not currently being used by CICS tasks.

enabled

File is enabled for access by CICS tasks.

disabled

File is disabled from access by CICS tasks.

Chapter 27. IccFileId class

IccBase
IccResourceId
IccFileId

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

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId

IccFileId

Method

nameLength
operator delete
operator new

Class

IccResourceId
IccBase
IccBase

Inherited protected methods

Method

operator=
setClassName
setCustomClassNum

Class

IccResourceId
IccBase
IccBase

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 **IccFile::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.

IccFileIterator

Conditions

DUPKEY, ENDFILE, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFIND, 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

Method	Class
<code>actionOnCondition</code>	<code>IccResource</code>
<code>actionOnConditionAsChar</code>	<code>IccResource</code>
<code>actionsOnConditionsText</code>	<code>IccResource</code>
<code>className</code>	<code>IccBase</code>
<code>classType</code>	<code>IccBase</code>
<code>condition</code>	<code>IccResource</code>
<code>conditionText</code>	<code>IccResource</code>
<code>customClassNum</code>	<code>IccBase</code>
<code>handleEvent</code>	<code>IccResource</code>
<code>id</code>	<code>IccResource</code>
<code>isEDFOn</code>	<code>IccResource</code>
<code>isRouteOptionOn</code>	<code>IccResource</code>
<code>name</code>	<code>IccResource</code>

Method

operator delete
operator new
routeOption
setActionOnAnyCondition
setActionOnCondition
setActionsOnConditions
setEDF
setRouteOption

Class

IccBase
IccBase
IccResource
IccResource
IccResource
IccResource
IccResource
IccResource

Inherited protected methods**Method**

setClassName
setCustomClassNum

Class

IccBase
IccBase

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

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

IccGroupId

Method
operator new

Class
IccBase

Inherited protected methods

Method
operator=
setClassName
setCustomClassNum

Class
IccResourceId
IccBase
IccBase

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.

lccJournal

If the current prefix was set by **setPrefix**, then the **lccJournal**'s copy of the buffer is deleted.

journalTypeId

const lccJournalTypeId& journalTypeId() const

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

put

virtual void put(const lccBuf& buffer)

buffer

A reference to an **lccBuf** object that holds data to be put into the journal.

A synonym for **writeRecord**—puts data into the journal. See “Polymorphic Behavior” on page 59 for information on polymorphism.

registerPrefix

void registerPrefix(const lccBuf* prefix)

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

setJournalTypeId (1)

void setJournalTypeId(const lccJournalTypeId& id)

setJournalTypeId (2)

void setJournalTypeId(const char* jtypeid)

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

setPrefix (1)

void setPrefix(const lccBuf& prefix)

setPrefix (2)

void setPrefix(const char* prefix)

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

wait

```
void wait (unsigned long requestNum=0,
          unsigned long option = 0)
```

requestNum

The write request. Zero indicates the last write on this journal.

option

An integer that affects the behaviour of **writeRecord** calls on the **IccJournal** object. Values other than 0 should be made from the **Options** enumeration, defined in this class. The values may be combined by addition or bitwise ORing, for example `IccJournal::startIO + IccJournal::synchronous`. The default is to use the system default.

Waits until a previous journal write has completed.

Condition: IOERR, JIDERR, NOTOPEN

writeRecord (1)

```
unsigned long writeRecord (const IccBuf& record,
                          unsigned long option = 0)
```

record

A reference to an **IccBuf** object that holds the record

option

See above.

writeRecord (2)

```
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 **wait** method in this class.

Conditions

IOERR, JIDERR, LENGERR, NOJBUFSP, NOTAUTH, NOTOPEN

Inherited public methods

Method	Class
<code>actionOnCondition</code>	<code>IccResource</code>
<code>actionOnConditionAsChar</code>	<code>IccResource</code>
<code>actionsOnConditionsText</code>	<code>IccResource</code>
<code>classType</code>	<code>IccBase</code>
<code>className</code>	<code>IccBase</code>
<code>condition</code>	<code>IccResource</code>
<code>conditionText</code>	<code>IccResource</code>
<code>customClassNum</code>	<code>IccBase</code>
<code>handleEvent</code>	<code>IccResource</code>

IccJournal

Method	Class
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

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.

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

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

Inherited protected methods

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

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: ICCRIDEH

`IccJournalTypeId` constructors

Constructor (1)

`IccJournalTypeId(const char* journalTypeName)`

journalTypeName

A 2-byte identifier used in journal records.

Constructor (2)

`IccJournalTypeId(const IccJournalId& id)`

id A reference to an `IccJournalTypeId` object.

Public methods

`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

Method	Class
<code>classType</code>	<code>IccBase</code>
<code>className</code>	<code>IccBase</code>
<code>customClassNum</code>	<code>IccBase</code>
<code>name</code>	<code>IccResourceId</code>
<code>nameLength</code>	<code>IccResourceId</code>
<code>operator delete</code>	<code>IccBase</code>

IccJournalTypeld

Method
operator new

Class
IccBase

Inherited protected methods

Method
operator=
setClassName
setCustomClassNum

Class
IccResourceId
IccBase
IccBase

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
```

lccKey

Returns an enumeration, defined in this class, that indicates whether the key is generic or complete.

operator= (1)

```
lccKey& operator=(const lccKey& key)
```

operator= (2)

```
lccKey& operator=(const lccBuf& buffer)
```

operator= (3)

```
lccKey& operator=(const char* value)
```

Assigns new value to key.

operator== (1)

```
lcc::Bool operator==(const lccKey& key) const
```

operator== (2)

```
lcc::Bool operator==(const lccBuf& text) const
```

operator== (3)

```
lcc::Bool operator==(const char* text) const
```

Tests equality.

operator!= (1)

```
lcc::Bool operator!=(const lccKey& key) const
```

operator!= (2)

```
lcc::Bool operator!=(const lccBuf& text) const
```

operator!= (3)

```
lcc::Bool operator!=(const char* text) const
```

Tests inequality.

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

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
length	IccRecordIndex
operator delete	IccBase
operator new	IccBase
type	IccRecordIndex
value	IccRecordIndex

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

Kind

complete

Specifies that the supplied key is not generic.

generic

Specifies that the search key is generic. A search is satisfied when a record is found with a key whose prefix matches the supplied key.

lccKey

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

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

IccLockId

Method
operator new

Class
IccBase

Inherited protected methods

Method
operator=
setClassName
setCustomClassNum

Class
IccResourceId
IccBase
IccBase

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.

IccMessage

summary

const char* summary()

Returns the text of the message.

text

const char* text() const

Returns the text of the message in the same way as summary.

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Chapter 36. IccPartnerId class

IccBase
IccResourceId
IccPartnerId

IccPartnerId class represents CICS remote (APPC) partner transaction definitions.

Header file: ICCRIDEH

IccPartnerId constructors

Constructor (1)

IccPartnerId(const char* partnerName)

partnerName

The 8-character name of an APPC partner.

Constructor (2)

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

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

IccPartnerId

Method
operator new

Class
IccBase

Inherited protected methods

Method
operator=
setClassName
setCustomClassNum

Class
IccResourceId
IccBase
IccBase

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 *abendCode* on page 71.

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.

lccProgram

If the current input message was set using **setInputMessage** then **clearInputMessage** releases the memory used by that buffer.

entryPoint

```
const void* entryPoint() const
```

Returns a pointer to the entry point of a loaded program module. This is only valid after a successful **load** call.

length

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

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

link

```
void link (const lccBuf* commArea = 0,  
           const lccTransId* transId = 0,  
           CommitOpt opt = noCommitOnReturn)
```

commArea

An optional pointer to the **lccBuf** 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 **lccTransId** object that indicates the name of the mirror transaction under which the program is to run if it is a remote (DPL) program link

opt

An enumeration, defined in this class, that affects the behavior of the link when the program is remote (DPL). The default (**noCommitOnReturn**) is not to commit resource changes on the remote CICS region until the current task commits its resources. The alternative (**commitOnReturn**) means that the resources of the remote program are committed whether or not this task subsequently abends or encounters a problem.

Conditions: INVREQ, NOTAUTH, PGMIDERR, SYSIDERR, LENGERR, ROLLEDBACK, TERMERR

Restrictions

Links may be nested, that is, a linked program may **link** to another program. However, due to implementation restrictions, you may only nest such programs 15 times. If this is exceeded, an exception is thrown.

load

```
void load(LoadOpt opt = releaseAtTaskEnd)
```

opt

An enumeration, defined in this class, that indicates whether CICS should automatically allow the program to be unloaded at task termination (releaseAtTaskEnd), or not (hold).

Conditions: NOTAUTH, PGMIDERR, INVREQ, LENGERR

registerInputMessage

void registerInputMessage(const IccBuf& msg)

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

setInputMessage

void setInputMessage(const IccBuf& msg)

Specifies data to be made available, by the **IccSession::receive()** method, to the called program, when using the **link** method in this class.

unload

void unload()

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

Conditions

NOTAUTH, PGMIDERR, INVREQ

Inherited public methods

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

CommitOpt

noCommitOnReturn

Changes to resources on the remote CICS region are not committed until the current task commits its resources. This is the default setting.

commitOnReturn

Changes to resources on the remote CICS region are committed whether or not the current task subsequently abends or encounters a problem.

LoadOpt

releaseAtTaskEnd

Indicates that CICS should automatically allow the program to be unloaded at task termination.

hold

Indicates that CICS should not automatically allow the program to be unloaded at task termination. (In this case, this or another task must explicitly use the **unload** method).

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

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase

IccProgramId

Method

name
nameLength
operator delete
operator new

Class

IccResourceId
IccResourceId
IccBase
IccBase

Inherited protected methods

Method

operator=
setClassName
setCustomClassNum

Class

IccResourceId
IccBase
IccBase

Chapter 39. lccRBA class

lccBase
lccRecordIndex
lccRBA

An **lccRBA** object holds a relative byte address which is used for accessing VSAM ESDS files.

Header file: ICCRECEH

lccRBA constructor

Constructor

lccRBA(unsigned long *initRBA* = 0)

initRBA

An initial value for the relative byte address.

Public methods

operator= (1)

lccRBA& operator=(const lccRBA& *rba*)

operator= (2)

lccRBA& operator=(unsigned long *num*)

num

A valid relative byte address.

Assigns a new value for the relative byte address.

operator== (1)

lcc::Bool operator== (const lccRBA& *rba*) const

operator== (2)

lcc::Bool operator== (unsigned long *num*) const

Tests equality

operator!= (1)

lcc!:Bool operator== (const lccRBA& *rba*) const

lccRBA

operator!= (2)

lcc::Bool operator!=(unsigned long *num*) const

Tests inequality

number

unsigned long number() const

Returns the relative byte address.

Inherited public methods

Method	Class
className	lccBase
classType	lccBase
customClassNum	lccBase
length	lccRecordIndex
operator delete	lccBase
operator new	lccBase
type	lccRecordIndex
value	lccRecordIndex

Inherited protected methods

Method	Class
setClassName	lccBase
setCustomClassNum	lccBase

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

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

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

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.

Inherited public methods

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

Inherited protected methods

Method	Class
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

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 177. 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 59 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 **IccResource** object and returns it as an **IccBuf** reference. This method is virtual and is implemented, wherever appropriate, in the derived classes. See Polymorphic Behavior on page 59 for a description of polymorphism. The default implementation in this class throws an exception to indicate that it has not been overridden in a subclass.

handleEvent

virtual HandleEventReturnOpt handleEvent(IccEvent& event)

event

A reference to an **IccEvent** object that describes the reason why this method is being called.

This virtual function may be re-implemented in a subclass (by the application programmer) to handle CICS events (see **IccEvent** class on page 123).

id

const IccResourceId* id() const

Returns a pointer to the **IccResourceId** object associated with this **IccResource** object.

isEDFOn

Icc::Bool isEDFOn() const

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

isRouteOptionOn

Icc::Bool isRouteOptionOn() const

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

name

const char* name() const

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

put

virtual void put(const IccBuf& buffer)

buffer

A reference to an **IccBuf** object that contains data that is to be put into the object.

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 59 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 50.

These methods force the object to route CICS requests to the named remote system. This is called explicit function shipping.

setRouteOption (1)

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

The parameters are:

sysId

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

setRouteOption (2)

```
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:

```
obj.setRouteOption()
```

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

ActionOnCondition

Possible values are:

noAction

Carry on as normal; it is the application programs 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** *typ*,
 const IccResourceId& *id*)

typ

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.

Protected methods

operator=

IccResourceId& operator=(const IccResourceId& id)

id

A reference to an **IccResourceId** object.

Set an **IccResourceId** object to be identical to *id*.

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Chapter 44. IccRRN class

IccBase
IccRecordIndex
IccRRN

An **IccRRN** object holds a relative record number and is used to identify records in VSAM RRDS files.

Header file: ICCRECEH

IccRRN constructors

Constructor

IccRRN(unsigned long *initRRN* = 1)

initRRN

The initial relative record number—an integer greater than 0. The default is 1.

Public methods

operator= (1)

IccRRN& operator=(const IccRRN& rrr)

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& rrr) const

operator== (2)

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

Tests equality

operator!= (1)

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

lccRRN

operator!= (2)

lcc::Bool operator!=(unsigned long num) const

Tests inequality

number

unsigned long number() const

Returns the relative record number.

Inherited public methods

Method	Class
className	IccBase
classType	IccBase
customClassNum	IccBase
length	IccRecordIndex
operator delete	IccBase
operator new	IccBase
type	IccRecordIndex
value	IccRecordIndex

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

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).

lock

lccSemaphore

void lock()

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

Conditions

ENQBUSY, LENGERR, INVREQ

tryLock

lcc::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

LockType type() const

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

unlock

void unlock()

Release a lock.

Conditions

LENGERR, INVREQ

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

LockType

byValue

The lock is on the contents (for example, name).

byAddress

The lock is on the memory address.

LifeTime

UOW The semaphore lasts for the length of the current unit of work.

task The semaphore lasts for the length of the task.

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.

Public methods

allocate

```
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)

```
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)

```
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)

```
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

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

const IccConvId& convId()

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

errorCode

const char* errorCode() const

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

extractProcess

void extractProcess()

Retrieves information from an APPC conversation attach header and holds it inside the object. See **PIPList**, **process**, and **syncLevel** methods to retrieve the information from the object. This method should be used by the back end task if it wants access to the PIP data, the process name, or the synclevel under which it is running.

Conditions

INVREQ, NOTALLOC, LENGERR

lccSession

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 lccBuf& get()

A synonym for **receive**. See “Polymorphic Behavior” on page 59 for information on polymorphism.

isErrorSet

lcc::Bool isErrorSet() const

Returns a boolean variable, defined in **lcc** structure, that indicates whether an error has been set.

isNoDataSet

lcc::Bool isNoDataSet() const

Returns a boolean variable, defined in **lcc** structure, that indicates if no data was returned on a **send**—just control information.

isSignalSet

lcc::Bool isSignalSet() const

Returns a boolean variable, defined in **lcc** 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

lccSession

const lccBuf& process() const

Returns a reference to an **lccBuf** 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 lccBuf& data)

data

A reference to an **lccBuf** object that holds the data to be sent to the remote process.

A synonym for **send**. See "Polymorphic Behavior" on page 59 for information on polymorphism.

receive

const lccBuf& receive()

Returns a reference to an **lccBuf** object that contains the data received from the remote system.

Conditions

EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

send (1)

**void send (const lccBuf& send,
SendOpt option = normal)**

send

A reference to an **lccBuf** 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.

option

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

sendInvite (2)

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

option

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

Sends data to the remote partner and indicates a change of direction, that is, the next method on this object will be **receive**.

Conditions

INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

sendLast (1)

```
void sendLast (const IccBuf& send,
               SendOpt option = normal)
```

send

A reference to an **IccBuf** object that contains the data that is to be sent.

option

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

sendLast (2)

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

option

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

Sends data to the remote partner and indicates that this is the final transmission. The **free** method must be invoked next, unless the sync level is 2, when you must commit resource updates before the **free**. (See **commitUOW** on page 215 in **IccTaskClass**).

Conditions

INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

IccSession

state

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
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

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase

Method	Class
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

AllocateOpt

queue

If all available sessions are in use, CICS is to queue this request (and block the method) until it can allocate a session.

noQueue

Control is returned to the application if it cannot allocate a session. CICS raises the SYSBUSY condition.

Indicates whether queuing is required on an **allocate** method.

SendOpt

normal

The default.

confirmation

Indicates that a program using SyncLevel level1 or level2 requires a response from the remote partner program. The remote partner can respond positively, using the **issueConfirmation** method, or negatively, using the **issueError** method. The sending program does not receive control back from CICS until the response is received.

wait

Requests that the data is sent and not buffered internally. CICS is free to buffer requests to improve performance if this option is not specified.

StateOpt

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.

IccSession

extractState

Return the explicitly extracted current state.

SyncLevel

level0

Sync level 0

level1

Sync level 1

level2

Sync level 2

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 204).

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**.

lccStartRequestQ

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

const lccBuf& data() const

Returns a reference to an **lccBuf** object that contains data passed on a start request. A call to this method should be preceded by a call to **retrieveData** method.

instance

static lccStartRequestQ* instance()

Returns a pointer to the single **lccStartRequestQ** object. If the object does not exist it is created. See also **startRequestQ** method on page 113 of **lccControl**.

queueName

const char* queueName() const

Returns the name of the queue that was passed by the start requester. A call to this method should be preceded by a call to **retrieveData** method.

registerData

void registerData(const lccBuf* buffer)

buffer

A pointer to the **lccBuf** object that holds data to be passed on a **start** request. Registers an **lccBuf** object to be interrogated for start data on each subsequent **start** method invocation.

This just stores the address of the **lccBuf** object within the **lccStartRequestQ** so that the **lccBuf** 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 **lccBuf** object during the time that it is invoked.

reset

void reset()

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

retrieveData

void retrieveData(RetrieveOpt option = noWait)

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 202 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)
```

IccStartRequestQ

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.

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

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource

IccStartRequestQ

Method	Class
setEDF	IccResource
setRouteOption	IccResource

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

RetrieveOpt

noWait
wait

ProtectOpt

none
protect

CheckOpt

check
noCheck

Chapter 48. IccSysId class

IccBase
IccResourceId
IccSysId

IccSysId class is used to identify a remote CICS system.

Header file: ICCRIDEH

IccSysId constructors

Constructor (1)

IccSysId(const char* name)

name

The 4-character name of the CICS system.

Constructor (2)

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

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

IccSysId

Method
operator new

Class
IccBase

Inherited protected methods

Method
operator=
setClassName
setCustomClassNum

Class
IccResourceId
IccBase
IccBase

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.

lccSystem

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

endBrowse

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 **lccSystem** **getStorage** method.

Conditions

INVREQ

getFile (1)

lccFile* getFile(const lccFileId& id)

id

A reference to an **lccFileId** object that identifies a CICS file.

getFile (2)

lccFile* getFile(const char* fileName)

fileName

The name of a CICS file.

Returns a pointer to the **lccFile** 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

lccSystem

operatingSystemLevel

unsigned short operatingSystemLevel()

Returns a halfword binary field giving the release number of the operating system under which CICS is running. The value returned is ten times the formal release number (the version number is not represented). For example, MVS/ESA Version 3 Release 2.1 would produce a value of 21.

Conditions

NOTAUTH

release

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

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

lccSysId& sysId()

Returns a reference to the **lccSysId** object that identifies this CICS system.

Conditions

INVREQ

workArea

const lccBuf& workArea()

Returns a reference to the **lccBuf** object that holds the work area for the CICS system.

Conditions

INVREQ

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

ResourceType

autoInstallModel
 connection
 dataQueue
 exitProgram
 externalDataSet
 file
 journal
 modename
 partner
 profile
 program
 requestId
 systemDumpCode
 tempStore
 terminal
 transactionDumpCode
 transaction
 transactionClass

Chapter 50. IccTask class

IccBase
IccResource
IccTask

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

Header file: ICCTSKEH

Sample: ICC\$TSK

IccTask Constructor (protected)

Constructor

IccTask()

Public methods

The opt parameter

Many methods have the same parameter, *opt*, which is described under the **abendCode** method in *abendCode* on page 71.

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

lccTask

void commitUOW()

Commit the resource updates within the current UOW for this task. This also causes a new UOW to start for subsequent resource update activity.

Conditions

INVREQ, ROLLEDBACK

delay

void delay (const lccTime& time, const lccRequestId* reqId = 0)

time

A reference to an object that contains information about the delay time. The object can be one of these types:

lccAbsTime

Expresses time as the number of milliseconds since the beginning of the year 1900.

lccTimeInterval

Expresses an interval of time, such as 3 hours, 2 minutes, and 1 second.

lccTimeOfDay

Expresses a time of day, such as 13 hours, 30 minutes (1-30 pm).

reqId

An optional pointer to an **lccRequestId** 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

const char* dump (const char* dumpCode, const lccBuf* buf = 0)

dumpCode

A 4-character label that identifies this dump

buf

A pointer to the **lccBuf** 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

```
void enterTrace (unsigned short traceNum,
                const char* resource = 0,
                lccBuf* 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 **lccBuf** 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

IccTask

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 211 in **IccSystem**class.

Conditions

LENGERR, NOSTG

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 166 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.

Conditions

INVREQ

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.

Conditions

INVREQ

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

Returns the priority for this task.

Conditions

INVREQ

rollBackUOW**void rollBackUOW()**

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

Conditions

INVREQ, ROLLEDBACK

setDumpOpts**void setDumpOpts(unsigned long opts = dDefault)**

lccTask

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.

Conditions

INVREQ

setWaitText

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

StartType startType()

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

Conditions

INVREQ

suspend

void suspend()

Suspend this task, allowing other tasks to be dispatched.

transId

const lccTransId& transId()

Returns the **lccTransId** object representing the transaction name of this CICS task.

triggerDataQueueId

const lccDataQueueId& 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

```
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

```
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

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

IccTask

id

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

Suspends the task until the alarm goes off (expires). See also **setAlarm** on page 101 in **IccClock**.

Conditions

INVREQ

workArea

IccBuf& workArea()

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

Conditions

INVREQ

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

AbendHandlerOpt

respectAbendHandler

Allows control to be passed to an abend handling program if one is in effect.

ignoreAbendHandler

Does not allow control to be passed to any abend handling program that may be in effect.

AbendDumpOpt**createDump**

Take a transaction dump when servicing an abend request.

suppressDump

Do not take a transaction dump when servicing an abend request.

DumpOpts

The values may be added, or bitwise ORed, together to get the desired combination. For example `IccTask::dProgram + IccTask::dDCT + IccTask::dSIT`.

dDefault**dComplete****dTask****dStorage****dProgram****dTerminal****dTables****dDCT****dFCT****dPCT****dPPT****dSIT****dTCT****dTRT****FacilityType**

none The task has no principal facility, that is, it is a background task.

terminal

This task has a terminal as its principal facility.

session

This task has a session as its principal facility, that is, it was probably started as a 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.

FEPIRequest

Front end programming interface. See *CICS/ESA: Front End Programming Interface User's Guide*, SC33-1175.

terminalInput

Started via a terminal input

CICSInternalTask

Started by CICS.

IccTask

StorageOpts

ifSOSReturnCondition

If insufficient space is available, return NOSTG condition instead of blocking the task.

below

Allocate storage below the 16Mb line.

userDataKey

Allocate storage in the USER data key.

CICSDataKey

Allocate storage in the CICS data key.

TraceOpt

normal

The trace entry is a standard entry.

exception

The trace entry is an exception entry.

WaitPostType

MVSPost

ECB is posted using the MVS POST service.

handPost

ECB is hand posted (that is, using some method other than the MVS POST service).

WaitPurgeability

purgeable

Task can be purged via a system call.

notPurgeable

Task cannot be purged via a system call.

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 in *abendCode* on page 71.

clear

virtual void clear()

A synonym for **empty**. See *Polymorphic Behavior* on page 59 for information on polymorphism.

lccTempStore

empty

void empty()

Deletes all the temporary data associated with the **lccTempStore** object and deletes the associated TD queue.

Conditions

INVREQ, ISCVREQ, NOTAUTH, QIDERR, SYSIDERR

get

virtual const lccBuf& get()

A synonym for **readNextItem**. See “Polymorphic Behavior” on page 59 for information on polymorphism.

numberOfItems

unsigned short numberOfItems() const

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

put

virtual void put(const lccBuf& buffer)

buffer

A reference to an **lccBuf** object that contains the data that is to be added to the end of the temporary storage queue.

A synonym for **writelnItem**. See “Polymorphic Behavior” on page 59 for information on polymorphism.

readItem

const lccBuf& 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 **lccBuf** object that contains the information.

Conditions

INVREQ, IOERR, ISCVREQ, ITEMERR, LENGERR, NOTAUTH, QIDERR, SYSIDERR

readNextItem

const lccBuf& readNextItem()

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

Conditions

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

rewriteltem

```
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)

```
unsigned short writeltem (const IccBuf& item,
                          NoSpaceOpt opt = suspend)
```

item

The name of the **IccBuf** object that contains the data that is to be 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)

```
unsigned short writeltem (const char* text,
                          NoSpaceOpt opt = suspend)
```

text

The text string that is to be 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, ISCVREQ, ITEMERR, LENGERR, NOSPACE, NOTAUTH, QIDERR, SYSIDERR

Inherited public methods

Method	Class
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

Location

auxStorage

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

memory

Temporary store data is to reside in memory.

NoSpaceOpt

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

suspend

Suspend the application program.

returnCondition

Do not suspend the application program, but raise the NOSPACE condition instead.

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

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase

IccTempStoreId

Method

name
nameLength
operator delete
operator new

Class

IccResourceId
IccResourceId
IccBase
IccBase

Inherited protected methods

Method

operator=
setClassName
setCustomClassNum

Class

IccResourceId
IccBase
IccBase

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

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase

IccTermId

Method

name
nameLength
operator delete
operator new

Class

IccResourceId
IccResourceId
IccBase
IccBase

Inherited protected methods

Method

operator=
setClassName
setCustomClassNum

Class

IccResourceId
IccBase
IccBase

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 71.

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 59 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

IccTerminal

IccTerminalData* data()

Returns a pointer to an **IccTerminalData** object that contains information about the characteristics of the terminal. The object is created if it does not already exist.

erase

void erase()

Erase all the data displayed at the terminal.

Conditions

INVREQ, INVPARTN

freeKeyboard

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 59 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)**IccTerminal& operator << (Color *color*)**

Sets the foreground color for data subsequently sent to the terminal.

operator<< (2)**IccTerminal& operator << (Highlight *highlight*)**

Sets the highlighting used for data subsequently sent to the terminal.

operator<< (3)**IccTerminal& operator << (const IccBuf& *buffer*)**

Writes another buffer.

operator<< (4)**IccTerminal& operator << (char *ch*)**

Writes a character.

operator<< (5)**IccTerminal& operator << (signed char *ch*)**

Writes a character.

operator<< (6)**IccTerminal& operator << (unsigned char *ch*)**

Writes a character.

lccTerminal

operator<< (7)

lccTerminal& operator << (const char* *text*)

Writes a string.

operator<< (8)

lccTerminal& operator << (const signed char* *text*)

Writes a string.

operator<< (9)

lccTerminal& operator << (const unsigned char* *text*)

Writes a string.

operator<< (10)

lccTerminal& operator << (short *num*)

Writes a short.

operator<< (11)

lccTerminal& operator << (unsigned short *num*)

Writes an unsigned short.

operator<< (12)

lccTerminal& operator << (long *num*)

Writes a long.

operator<< (13)

lccTerminal& operator << (unsigned long *num*)

Writes an unsigned long.

operator<< (14)

lccTerminal& operator << (int *num*)

Writes an integer.

operator<< (15)

lccTerminal& operator << (float *num*)

Writes a float.

operator<< (16)

IccTerminal& operator << (double num)

Writes a double.

operator<< (17)

IccTerminal& operator << (long double num)

Writes a long double.

operator<< (18)

IccTerminal& operator << (IccTerminal& (*f)(IccTerminal&))

Enables the following syntax:

```
Term << Hello World << endl;
Term << Hello again << flush;
```

put

virtual void put(const IccBuf& buf)

A synonym for **sendLine**. See “Polymorphic Behavior” on page 59 for information on polymorphism.

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

lccTerminal

send (1)

```
void send(const lccBuf& buffer)
```

buffer

A reference to an **lccBuf** 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 lccBuf& 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 **lccBuf** 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*.

Writes the specified data to either the current cursor position or to the cursor position specified by the arguments.

Conditions

INVREQ, LENGERR, TERMERR

send3270Data (1)

```
void send3270Data(const IccBuf& buffer)
```

buffer

A reference to an **IccBuf** object that holds the data that is to be sent.

send3270Data (2)

```
void send3270 Data(const char* format,
                  ...)
```

format

A format string, as in the **printf** standard library function

...

The optional arguments that accompany *format*.

send3270Data (3)

```
void send3270Data (unsigned short col,
                  const IccBuf& buf)
```

col

The column where the writing of the data is started

buffer

A reference to an **IccBuf** object that holds the data that is to be sent.

send3270Data (4)

```
void send3270Data (unsigned short col,
                  const char* format,
                  ...)
```

col

The column where the writing of the data is started

format

A format string, as in the **printf** standard library function

...

The optional arguments that accompany *format*.

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)

```
void sendLine(const IccBuf& buffer)
```

buffer

A reference to an **IccBuf** object that holds the data that is to be sent.

IccTerminal

sendLine (2)

```
void sendLine (const char* format,  
              ...)
```

format

A format string, as in the **printf** standard library function

...

The optional arguments that accompany *format*.

sendLine (3)

```
void sendLine (unsigned short col,  
              const IccBuf& buf)
```

col

The column where the writing of the data is started

buffer

A reference to an **IccBuf** object that holds the data that is to be sent.

sendLine (4)

```
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

```
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)

```
void setCursor(unsigned short offset)
```

offset

The position of the cursor where the top left corner is 0.

setCursor (2)

```
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

```
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

```
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

```
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

```
void setNextCommArea(const IccBuf& commArea)
```

commArea

A reference to the buffer that is to be used as a COMMAREA.

IccTerminal

Specifies the COMMAREA that is to be passed to the next transaction started on this terminal.

setNextInputMessage

```
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

```
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

```
void signoff()
```

Signs off the user who is currently signed on. Authority reverts to the default user.

Conditions

INVREQ

signon (1)

```
void signon (const IccUserId& id,  
            const char* password = 0,  
            const char* newPassword = 0)
```

id

A reference to an **IccUserId** object

password

The 8-character existing password.

newPassword

An optional 8-character new password.

signon (2)

```
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

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

Enumerations

AIDVal

ENTER
CLEAR
PA1 to PA3
PF1 to PF24

Case

upper
mixed

Color

defaultColor
blue
red
pink
green
cyan
yellow
neutral

Highlight

defaultHighlight
blink
reverse
underscore

NextTransIdOpt

queue

Queue the transaction with any other outstanding starts queued on the terminal.

immediate

Start the transaction immediately, that is, before any other outstanding starts queued on the terminal.

Chapter 55. IccTerminalData class

IccBase
IccResource
IccTerminalData

IccTerminalData is a singleton class owned by **IccTerminal** (see **data** on page 233 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 71.

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

lccTerminalData

defaultHeight

unsigned short defaultHeight(lcc::GetOpt *opt* = lcc::object)

Returns the default height of the screen, in lines.

Conditions

INVREQ

defaultWidth

unsigned short defaultWidth(lcc::GetOpt *opt* = lcc::object)

Returns the default width of the screen, in characters.

Conditions

INVREQ

graphicCharCodeSet

unsigned short graphicCharCodeSet(lcc::GetOpt *opt* = lcc::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

unsigned short graphicCharSetId(lcc::GetOpt *opt* = lcc::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

lcc::Bool isAPLKeyboard(lcc::GetOpt *opt* = lcc::object)

Returns a boolean that indicates whether the terminal has the APL keyboard feature.

Conditions

INVREQ

isAPLText

lcc::Bool isAPLText(lcc::GetOpt *opt* = lcc::object)

Returns a boolean that indicates whether the terminal has the APL text feature.

Conditions

INVREQ

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

Returns a boolean that indicates whether the terminal has the background transparency capability.

Conditions

INVREQ

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

Returns a boolean that indicates whether the terminal has the extended color capability.

Conditions

INVREQ

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

Returns a Boolean that indicates whether the terminal supports Erase Write Alternative.

Conditions

INVREQ

isExtended3270**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**Icc::Bool isFieldOutline(Icc::GetOpt *opt* = Icc::object)**

Returns a boolean that indicates whether the terminal supports field outlining.

Conditions

INVREQ

lccTerminalData

isGoodMorning

lcc::Bool isGoodMorning(lcc::GetOpt opt = lcc::object)

Returns a boolean that indicates whether the terminal has a good morning message.

Conditions

INVREQ

isHighlight

lcc::Bool isHighlight(lcc::GetOpt opt = lcc::object)

Returns a boolean that indicates whether the terminal has extended highlight capability.

Conditions

INVREQ

isKatakana

lcc::Bool isKatakana(lcc::GetOpt opt = lcc::object)

Returns a boolean that indicates whether the terminal supports Katakana.

Conditions

INVREQ

isMSRControl

lcc::Bool isMSRControl(lcc::GetOpt opt = lcc::object)

Returns a boolean that indicates whether the terminal supports magnetic slot reader control.

Conditions

INVREQ

isPS

lcc::Bool isPS(lcc::GetOpt opt = lcc::object)

Returns a boolean that indicates whether the terminal supports programmed symbols.

Conditions

INVREQ

isSOSI

lcc::Bool isSOSI(lcc::GetOpt opt = lcc::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

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

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.

lccTime

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 **lccTime** this is.

Inherited public methods

Method	Class
actionOnCondition	lccResource
actionOnConditionAsChar	lccResource
actionsOnConditionsText	lccResource
className	lccBase
classType	lccBase
condition	lccResource
conditionText	lccResource
customClassNum	lccBase
handleEvent	lccResource
isEDFOn	lccResource
operator delete	lccBase
operator new	lccBase
setActionOnAnyCondition	lccResource
setActionOnCondition	lccResource
setActionsOnConditions	lccResource
setEDF	lccResource

Inherited protected methods

Method	Class
setClassName	lccBase
setCustomClassNum	lccBase

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

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

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

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

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

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

IccTPNameId

Method
operator new

Class
IccBase

Inherited protected methods

Method
operator=
setClassName
setCustomClassNum

Class
IccResourceId
IccBase
IccBase

Chapter 60. IccTransId class

IccBase
IccResourceId
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

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase

IccTransId

Method

name
nameLength
operator delete
operator new

Class

IccResourceId
IccResourceId
IccBase
IccBase

Inherited protected methods

Method

operator=
setClassName
setCustomClassNum

Class

IccResourceId
IccBase
IccBase

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 users group ID.

Constructor (2)

```
IccUser (const char* userName,  
         const char* groupName = 0)
```

userName

The 8-character user ID

gid

The optional 8-character group ID.

Public methods

changePassword

```
void changePassword (const char* password,  
                    const char* newPassword)
```

password

The users existing password—a string of up to 8 characters

newPassword

The users 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 users group ID.

invalidPasswordAttempts

unsigned long invalidPasswordAttempts() const

Returns the number of times the wrong password has been entered for this user since the last successful signon. This method should only be used after a successful **verifyPassword** method.

language

const char* language() const

Returns the user's language after a successful call to **signon** in **iccTerminal**.

lastPasswordChange

const iccAbsTime& lastPasswordChange() const

Returns a reference to an **iccAbsTime** object that holds the time when the password was last changed. This method should only be used after a successful **verifyPassword** method.

lastUseTime

const IccAbsTime& lastUseTime() const

Returns a reference to an **IccAbsTime** object that holds the time when the user ID was last used. This method should only be used after a successful **verifyPassword** method.

passwordExpiration

const IccAbsTime& passwordExpiration() const

Returns a reference to an **IccAbsTime** object that holds the time when the password will expire. This method should only be used after a successful **verifyPassword** method.

setLanguage

void setLanguage(const char* language)

Sets the IBM-defined national language code that is to be associated with this user. This should be a three character value.

verifyPassword

void verifyPassword(const char* password)

Checks that the supplied password matches the password recorded by the external security manager for this **IccUser**.

Conditions

INVREQ, NOTAUTH, USERIDERR

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase
setCustomClassNum	IccBase

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

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

IccUserId

Method
operator new

Class
IccBase

Inherited protected methods

Method
operator=
setClassName
setCustomClassNum

Class
IccResourceId
IccBase
IccBase

Chapter 63. IccValue structure

This structure contains CICS-value data areas (CVDAs) as an enumeration.

Header file: ICCVALEH

Enumeration

CVDA

Valid CVDAs are:

ACQFAIL	ACQUIRED	ACQUIRING	ACTIVE
ADD	ADDABLE	ADDFAIL	ADVANCE
ALARM	ALLCONN	ALLOCATED	ALLQUERY
ALTERABLE	ALTERNATE	ALTPRTCOPY	ANY
APLKYBD	APLTEXT	APPC	APPCPARALLEL
APPCSINGLE	APPLICATION	ASACTL	ASCII7
ASCII8	ASSEMBLER	ATI	ATTENTION
AUDALARM	AUTOACTIVE	AUTOARCH	AUTOCONN
AUTOINACTIVE	AUTOPAGEABLE	AUTOSTART	AUXILIARY
AUXPAUSE	AUXSTART	AUXSTOP	AVAILABLE
BACKOUT	BACKTRANS	BACKUPNONBWO	BASE
BASESPACE	BATCHLU	BDAM	BEGINSESSION
BELOW	BGAM	BIPROG	BISYNCH
BLK	BLOCKED	BROWSABLE	BSAM
	BUSY	C	CACHE
CANCEL	CANCELLED	CD	CDRDLPRT
CEDF	CICS	CICSDATAKEY	CICSEXECKEY
CICSSECURITY	CICSTABLE	CLEAR	CLOSED
CLOSEFAILED	CLOSELEAVE	CLOSEREQUEST	CLOSING
CMDPROT	CMDSECEXT	CMDSECNO	CMDSECYES
COBOL	COBOLII	COLDACQ	COLDQUERY
COLDSTART	COLOR	COMMIT	COMMITFAIL
CONFFREE	CONFRECEIVE	CONFSEND	CONNECTED
CONNECTION	CONSISTENT	CONSOLE	CONTNLU
CONTROLSHUT	CONVERSE	CONVIDLE	COORDINATOR
COPY	CREATE	CRITICAL	CTLGALL
CTLGMODIFY	CTLGNONE	CTRLABLE	CURRENT
DAE	DATA	DATASET	DATASETFULL
DATASTREAM	DB2®	DEADLOCK	DEC
DEFAULT	DEFRESP1	DEFRESP1OR2	DEFRESP2
DEFRESP3	DELAY	DELETABLE	DELETEFAIL
DELEXITERROR	DEREGERROR	DEREGISTERED	DEST
DISABLED	DISABLING	DISCARDFAIL	DISCREQ
DISK1	DISK2	DISK2PAUSE	DISPATCHABLE
DPLSUBSET	DS3270	DUALCASE	DUMMY
DYNAMIC	EB	EMERGENCY	EMPTY
EMPTYREQ	ENABLED	ENDAFFINITY	ESDS
EVENT	EVENTUAL	EXCEPT	EXCEPTRESP
EXCI	EXCTL	EXECENQ	EXECENQADDR
EXITTRACE	EXTENDEDSDS	EXTRA	EXTSECURITY
FAILED	FAILEDDBKOUT	FAILINGBKOUT	FCLOSE
FILE	FINALQUIESCE	FINPUT	FIRSTINIT

IccValue

FIRSTQUIESCE	FIXED	FLUSH	FMH
FMHPARM	FOPEN	FORCE	FORCECANCEL
FORCECLOSE	FORCECLOSING	FORCEPURGE	FORCEUOW
FORMATEDF	FORMATTED	FORMFEED	FOUTPUT
FREE	FREEING	FULL	FULLAPI
FWDRECOVABLE	GENERIC	GMT	GOINGOUT
GTFSTART	GTFSTOP	HARDCOPY	HEURBACKOUT
HEURCOMMIT	HEX	HFORM	HILIGHT
HOLD	IGNORE	IGNORERR	IMMCLOSE
IMMCLOSING	IMMEDIATE	IMMQUIESCED	INACTIVE
INBOUND	INDEXRECFULL	INDIRECT	INDOUBT
INFLIGHT	INITCOMPLETE	INOUT	INPUT
INSERVICE	INSTALLED	INSTALLFAIL	INTACTLU
INTRA	INTSTART	INTSTOP	INVALID
IOERROR	IRC	ISCMCONV	ISOLATE
KATAKANA	KEYED	KSDS	LE370
LEAVE	LIC	LIGHTPEN	LOCAL
LOG	LOGICAL	LOGTERM	LOSE
LPA	LU61	LUCMODGRP	LUCSESS
LUP	LUSTAT	LUTYPE4	LUTYPE6
LUW	MAGTAPE	MAIN	MAP
MAPSET	MCHCTL	MDT	MOD
MODE24	MODE31	MODEANY	MODEL
MORE	MSRCONTROL	MVS	NEGATIVE
NEW	NEWCOPY	NEWSESSION	NOALARM
NOALPRTCOPY	NOAPLKYBD	NOAPLTEXT	NOATI
NOAUDALARM	NOAUTOARCH	NOBACKOUT	NOBACKTRANS
NOCEDF	NOCLEAR	NOCMDPROT	NOCOLOR
NOCONV	NOCONVERSE	NOCOPY	NOCREATE
NOCTL	NODAE	NODISCREQ	NODUALCASE
NOEMPTYREQ	NOEVENT	NOEXCEPT	NOEXCTL
NOEXITTRACE	NOEXTENDEDDES	NOFMH	NOFMHPARM
NOFORMATEDF	NOFORMFEED	NOHFORM	NOHILIGHT
NOHOLD	NOISOLATE	NOKATAKANA	NOLIGHTPEN
NOLOG	NOLOSTLOCKS	NOMDT	NOMSGJRNL
NOMSRCONTROL	NONAUTOCONN	NONCICS	NONE
NOOBFORMAT	NOOOPERID	NOOUTLINE	NOPARTITIONS
NOPERF	NOPRESETSEC	NOPRINTADAPT	NOPROGSYMBOL
NOPRTCOPY	NOQUERY	NORECOVDATA	NOREENTPROT
NORELREQ	NORETAINED	NORMALBKOUT	NORMALRESP
NOSECURITY	NOSHUTDOWN	NOSOSI	NOSPI
NOSTSN	NOSWITCH	NOSYNCPPOINT	NOSYSDUMP
NOSYSLOG	NOTADDABLE	NOTALTERABLE	NOTAPPLIC
NOTASKSTART	NOTBROWSABLE	NOTBUSY	NOTCDEB
NOTCONNECTED	NOTCTRLABLE	NOTDEFINED	NOTDELETABLE
NOTEMPTY	NOTERMINAL	NOTEXTKYBD	NOTEXTPRINT
NOTFWDRCVBLE	NOTINBOUND	NOTINIT	NOTINSTALLED
NOTKEYED	NOTLPA	NOTPENDING	NOTPURGEABLE
NOTRANDUMP	NOTREADABLE	NOTREADY	NOTRECOVABLE
NOTREQUIRED	NOTRLS	NOTSOS	NOTSUPPORTED
NOTTABLE	NOTTI	NOTUPDATABLE	NOUCTRAN
NOVALIDATION	NOVFORM	NOWAIT	NOWRITE
NOZCPTRACE	OBFORMAT	OBOPERID	OBTAINING
OFF	OK	OLD	OLDCOPY
OLDSESSION	ON	OPEN	OPENERROR

OPENING	OPENINPUT	OPENOUTPUT	OUTLINE
OUTPUT	OUTSERVICE	OWNER	PAGEABLE
PARTITIONS	PARTITIONSET	PATH	PENDBEGIN
PENDDATA	PENDFREE	PENDING	PENDPASS
PENDRECEIVE	PENDRELEASE	PENDSTART	PENDSTSN
PENDUNSOL	PERF	PHASEIN	PHYSICAL
PL1	PLI	POSITIVE	POST
PRESETSEC	PRIMARY	PRINTADAPT	PRIVATE
PROFILE	PROGRAM	PROGSYMBOL	PROTECTED
PRTCOPY	PURGE	PURGEABLE	QUEUE
QUIESCED	QUIESCING	READABLE	READBACK
READONLY	READY	RECEIVE	RECOVDATA
RECOVERABLE	RECOVERED	RECOVERLOCKS	REENTPROT
REGERROR	REGISTERED	REJECT	RELATED
RELEASE	RELEASED	RELEASING	RELREQ
REMLOSTLOCKS	REMOTE	REMOVE	REMSSESSION
REPEATABLE	REQUIRED	RERED	RESET
RESETLOCKS	RESSECEXT	RESSECNO	RESSECYES
RESSYS	RESYNC	RETAINED	RETRY
REVERTED	RLS	RLSACTIVE	RLSGONE
RLSINACTIVE	RLSSERVER	RMI	ROLLBACK
ROUTE	RPG	RRDS	RTR
RU	RUNNING	SCS	SDLC
SECONDINIT	SEND	SEQDISK	SESSION
SESSIONFAIL	SESSIONLOST	SETFAIL	SFS
SHARE	SHARED	SHUNTED	SHUTDISABLED
SHUTDOWN	SHUTENABLED	SIGNEDOFF	SIGNEDON
SINGLEOFF	SINGLEON	SKIP	SMF
SNA	SOS	SOSABOVE	SOSBELOW
SOSI	SPECIFIC	SPECTRACE	SPI
SPRSTRACE	STANDBY	STANTRACE	START
STARTED	STARTING	STARTUP	STATIC
STOPPED	STSN	STSNSET	STSNTEST
SUBORDINATE	SUBSPACE	SURROGATE	SUSPENDED
SWITCH	SWITCHALL	SWITCHING	SWITCHNEXT
SYNCFREE	SYNCPOINT	SYNCRECEIVE	SYNCSND
SYS370	SYS7BSCA	SYSDUMP	SYSLOG
SYSTEM3	SYSTEM	SYSTEM7	SYSTEMOFF
SYSTEMON	T1050	T1053	T2260L
T2260R	T2265	T2740	T2741BCD
T2741COR	T2770	T2780	T2980
T3275R	T3277L	T3277R	T3278M2
T3278M3	T3278M4	T3278M5	T3279M2
T3279M3	T3279M4	T3279M5	T3284L
T3284R	T3286L	T3286R	T3600BI
T3601	T3614	T3650ATT	T3650PIPE
T3650USER	T3653HOST	T3735	T3740
T3780	T3790	T3790SCSP	T3790UP
T7770	TAKEOVER	TAPE1	TAPE2
TASK	TASKSTART	TCAM	TCAMSNA
TCEXITALL	TCEXITALLOFF	TCEXITNONE	TCEXITSYSTEM
TCLASS	TCONSOLE	TDQ	TELETYPE
TERM	TERMINAL	TEXTKYBD	TEXTPRINT
THIRDINIT	TIME	TIMEOUT	TPS55M2
TPS55M3	TPS55M4	TPS55M5	TRANDUMP

IccValue

TRANIDONLY	TSQ	TTCAM	TTI
TWX3335	UCTRAN	UNAVAILABLE	UNBLOCKED
UNCOMMITTED	UNCONNECTED	UNDEFINED	UNDETERMINED
UNENABLED	UNENABLING	UNKNOWN	UNPROTECTED
UNQUIESCED	UNREGISTERED	UNSOLDATA	UOW
UPDATABLE	USER	USERDATAKEY	USEREXECKEY
USEROFF	USERON	USERTABLE	VALID
VALIDATION	VARIABLE	VFORM	VIDEOTERM
VRRDS	VSAM	VTAM®	WAIT
WAITCOMMIT	WAITER	WAITFORGET	WAITING
WAITRMI	WARMSTART	WIN	XCF
XM	XNOTDONE	XOK	ZCPTRACE

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 61)
 - Family Subset enforcement (see page 70)
 - EDF enablement (see page 50)
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.

```
# int main() 1
#
# {
#     Icc::initializeEnvironment(ICC_CLASS_MEMORY_MGMT, 2
#                               ICC_FAMILY_SUBSET,
#                               ICC_EDF_BOOL);
#
#     try 3
#     {
#         ICC_USER_CONTROL control; 4
#         control.run(); 5
#     }
#     catch(IccException& exc) 6
#     {
#         Icc::catchException(exc); 7
#     }
#     catch(...) 8
#     {
#         Icc::unknownException(); 9
#     }
#     Icc::returnToCICS(); 10
# }
```

1 This is the main C++ entry point.

2 This call initializes the environment and is essential. The three parameters have previously been defined to the defaults for the platform.

main function

- 3 Run the users 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.

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

EXEC CICS to Foundation Class methods

EXEC CICS	Class	Method
ASSIGN GMMI	IccTerminalData	isGoodMorning
ASSIGN HIGHLIGHT	IccTerminalData	isHighlight
ASSIGN INITPARM	IccControl	initData
ASSIGN INITPARMLEN	IccControl	initData
ASSIGN INVOKINGPROG	IccControl	callingProgramId
ASSIGN KATAKANA	IccTerminalData	isKatakana
ASSIGN NETNAME	IccTerminal	netName
ASSIGN OUTLINE	IccTerminalData	isFieldOutline
ASSIGN ORGABCODE	IccAbendData	originalAbendCode
ASSIGN PRINSYSID	IccTask	principalSysId
ASSIGN PROGRAM	IccControl	programId
ASSIGN PS	IccTerminalData	isPS
ASSIGN QNAME	IccTask	triggerDataQueueId
ASSIGN RESSEC	IccTask	isResourceSecurityOn
ASSIGN RESTART	IccTask	isRestarted
ASSIGN SCRNH	IccTerminal	height
ASSIGN SCRWD	IccTerminal	width
ASSIGN SOSI	IccTerminalData	isSOSI
ASSIGN STARTCODE	IccTask	startType, isCommitSupported, isStartDataAvailable
ASSIGN SYSID	IccSystem	sysId
ASSIGN TASKPRIORITY	IccTask	priority
ASSIGN TCTUALENG	IccTerminal	workArea
ASSIGN TEXTKYBD	IccTerminalData	isTextKeyboard
ASSIGN TEXTPRINT	IccTerminalData	isTextPrint
ASSIGN TWALENG	IccTask	workArea
ASSIGN USERID	IccTask	userId
ASSIGN VALIDATION	IccTerminalData	isValidation
CANCEL	IccClock	cancelAlarm
CANCEL	IccStartRequestQ	cancel
CHANGE PASSWORD	IccUser	changePassword
CHANGE TASK	IccTask	setPriority
CONNECT PROCESS	IccSession	connectProcess
CONVERSE	IccSession	converse
DELAY	IccTask	delay
DELETE	IccFile	deleteRecord
DELETE	IccFile	deleteLockedRecord
DELETEQ TD	IccDataQueue	empty
DELETEQ TS	IccTempStore	empty
DEQ	IccSemaphore	unlock

EXEC CICS to Foundation Class methods

EXEC CICS	Class	Method
DUMP TRANSACTION	IccTask	dump
DUMP TRANSACTION	IccTask	setDumpOpts
ENDBR	IccFileIterator	IccFileIterator (destructor)
ENQ	IccSemaphore	lock
ENQ	IccSemaphore	tryLock
ENTER TRACENUM	IccTask	enterTrace
EXTRACT ATTRIBUTES	IccSession	state, stateText
EXTRACT PROCESS	IccSession	extractProcess
FORMATTIME YYDDD, YYMMDD, etc	IccClock	date
FORMATTIME DATE	IccClock	date
FORMATTIME DATEFORM	IccSystem	dateFormat
FORMATTIME DAYCOUNT	IccClock	daysSince1900
FORMATTIME DAYOFWEEK	IccClock	dayOfWeek
FORMATTIME DAYOFMONTH	IccClock	dayOfMonth
FORMATTIME MONTHOFYEAR	IccClock	monthOfYear
FORMATTIME TIME	IccClock	time
FORMATTIME YEAR	IccClock	year
FREE	IccSession	free
FREEMAIN	IccTask	freeStorage
GETMAIN	IccTask	getStorage
HANDLE ABEND	IccControl	setAbendHandler, cancelAbendHandler, resetAbendHandler
INQUIRE FILE ACCESSMETHOD	IccFile	accessMethod
INQUIRE FILE ADD	IccFile	isAddable
INQUIRE FILE BROWSE	IccFile	isBrowsable
INQUIRE FILE DELETE	IccFileControl	isDeletable
INQUIRE FILE EMPTYSTATUS	IccFile	isEmptyOn
INQUIRE FILE ENABLESTATUS	IccFile	enableStatus
INQUIRE FILE KEYPOSITION	IccFile	keyPosition
INQUIRE FILE OPENSTATUS	IccFile	openStatus
INQUIRE FILE READ	IccFile	isReadable
INQUIRE FILE RECORDFORMAT	IccFile	recordFormat
INQUIRE FILE RECORDSIZE	IccFile	recordLength

EXEC CICS to Foundation Class methods

EXEC CICS	Class	Method
INQUIRE FILE RECOVSTATUS	IccFile	isRecoverable
INQUIRE FILE TYPE	IccFile	type
INQUIRE FILE UPDATE	IccFile	isUpdatable
ISSUE ABEND	IccSession	issueAbend
ISSUE CONFIRMATION	IccSession	issueConfirmation
ISSUE ERROR	IccSession	issueError
ISSUE PREPARE	IccSession	issuePrepare
ISSUE SIGNAL	IccSession	issueSignal
LINK	IccProgram	link
LINK INPUTMSG INPUTMSGLEN	IccProgram	setInputMessage
LOAD	IccProgram	load
POST	IccClock	setAlarm
READ	IccFile	readRecord
READNEXT	IccFileIterator	readNextRecord
READPREV	IccFileIterator	readPreviousRecord
READQ TD	IccDataQueue	readItem
READQ TS	IccTempStore	readItem
RECEIVE (APPC)	IccSession	receive
RECEIVE (3270)	IccTerminal	receive, receive3270Data
RELEASE	IccProgram	unload
RESETBR	IccFileIterator	reset
RETRIEVE	IccStartRequestQ	retrieveData ¹
<p>Note: The retrieveData method gets the start information from CICS and stores it in the IccStartRequestQ object: the information can then be accessed using data, queueName, returnTermId and returnTransId methods.</p>		
RETRIEVE INTO, LENGTH	IccStartRequestQ	data
RETRIEVE QUEUE	IccStartRequestQ	queueName
RETRIEVE RTRANSID	IccStartRequestQ	returnTransId
RETRIEVE RTERMID	IccStartRequestQ	returnTermId
RETURN	IccControl	main ²
<p>Note: Returning (using C++ reserved word return) from method run in class IccControl results in an EXEC CICS RETURN.</p>		
RETURN TRANSID	IccTerminal	setNextTransId ³
RETURN IMMEDIATE	IccTerminal	setNextTransId ³
RETURN COMMAREA LENGTH	IccTerminal	setNextCommArea ³
RETURN INPUTMSG, INPUTMSGLEN	IccTerminal	setNextInputMessage ³
<p>Note: Issue this call before returning from IccControl::run.</p>		
REWRITE	IccFile	rewriteRecord
SEND (APPC)	IccSession	send, sendInvite, sendLast

EXEC CICS to Foundation Class methods

EXEC CICS	Class	Method
SEND (3270)	IccTerminal	send, sendLine
SEND CONTROL CURSOR	IccTerminal	setCursor setLine, setNewLine
SEND CONTROL ERASE	IccTerminal	erase
SEND CONTROL FREEKB	IccTerminal	freeKeyboard
SET FILE ADDIBROWSEIDELETEI...	IccFile	setAccess
SET FILE EMPTYSTATUS	IccFile	setEmptyOnOpen
SET FILE OPEN STATUSIENABLESTATUS	IccFile	setStatus
SIGNOFF	IccTerminal	signoff
SIGNON	IccTerminal	signon
START TRANSID AT/AFTER	IccStartRequestQ	start ⁴
START TRANSID FROM LENGTH	IccStartRequestQ	setData, registerDataBuffer ⁴
START TRANSID NOCHECK	IccStartRequestQ	setStartOpts ⁴
START TRANSID PROTECT	IccStartRequestQ	setStartOpts ⁴
START TRANSID QUEUE	IccStartRequestQ	setQueueName ⁴
START TRANSID REQID	IccStartRequestQ	start ⁴
START TRANSID TERMID	IccStartRequestQ	start ⁴
START TRANSID USERID	IccStartRequestQ	start ⁴
START TRANSID RTERMID	IccStartRequestQ	setReturnTermId ⁴
START TRANSID RTRANSID	IccStartRequestQ	setReturnTransId ⁴
Note: Use methods setData , setQueueName , setReturnTermId , setReturnTransId , setStartOpts to set the state of the IccStartRequestQ object before issuing start requests with the start method.		
STARTBR	IccFileIterator	IccFileIterator (constructor)
SUSPEND	IccTask	suspend
SYNCPOINT	IccTask	commitUOW
SYNCPOINT ROLLBACK	IccTask	rollBackUOW
UNLOCK	IccFile	unlockRecord
VERIFY PASSWORD	IccUser	verifyPassword
WAIT CONVID	IccSession	flush
WAIT EVENT	IccTask	waitOnAlarm
WAIT EXTERNAL	IccTask	waitExternal
WAIT JOURNALNUM	IccJournal	wait
WRITE	IccFile	writeRecord
WRITE OPERATOR	IccConsole	write, writeAndGetReply
WRITEQ TD	IccDataQueue	writeltem
WRITEQ TS	IccTempStore	writeltem, rewriteltem

Appendix B. Mapping Foundation Class methods to EXEC CICS calls

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

IccAbendData Class	
Method	EXEC CICS
abendCode	ASSIGN ABCODE
ASRAInterrupt	ASSIGN ASRAINTRPT
ASRAKeyType	ASSIGN ASRAKEY
ASRAPSW	ASSIGN ASRAPSW
ASRARegisters	ASSIGN ASRAREGS
ASRASpaceType	ASSIGN ASRASPC
ASRAStorageType	ASSIGN ASRASTG
isDumpAvailable	ASSIGN ABDUMP
originalAbendCode	ASSIGN ORGABCODE
programName	ASSIGN ABPROGRAM
IccAbsTime Class	
Method	EXEC CICS
date	FORMATTIME YYDDD/YYMMDD/etc.
dayOfMonth	FORMATTIME DAYOFMONTH
dayOfWeek	FORMATTIME DAYOFWEEK
daysSince1900	FORMATTIME DAYCOUNT
monthOfYear	FORMATTIME MONTHOFYEAR
time	FORMATTIME TIME
year	FORMATTIME YEAR
IccClock Class	
Method	EXEC CICS
cancelAlarm	CANCEL
date	FORMATTIME YYDDD/YYMMDD/etc.
dayOfMonth	FORMATTIME DAYOFMONTH
dayOfWeek	FORMATTIME DAYOFWEEK
daysSince1900	FORMATTIME DAYCOUNT
monthOfYear	FORMATTIME MONTHOFYEAR
setAlarm	POST
time	FORMATTIME TIME
update	ASKTIME
year	FORMATTIME YEAR
IccConsole Class	
Method	EXEC CICS
write	WRITE OPERATOR

Foundation Class methods to EXEC CICS

writeAndGetReply	WRITE OPERATOR
IccControl Class	
Method	EXEC CICS
callingProgramId	ASSIGN INVOKINGPROG
cancelAbendHandler	HANDLE ABEND CANCEL
commArea	ADDRESS COMMAREA
initData	ASSIGN INITPARM & INITPARMLEN
programId	ASSIGN PROGRAM
resetAbendHandler	HANDLE ABEND RESET
setAbendHandler	HANDLE ABEND PROGRAM
IccDataQueue Class	
Method	EXEC CICS
empty	DELETEQ TD
readItem	READQ TD
writeItem	WRITEQ TD
IccFile Class	
Method	EXEC CICS
access	INQUIRE FILE ADDIBROWSEDELETEIREADIUPDATE
accessMethod	INQUIRE FILE ACCESSMETHOD
deleteRecord	DELETE FILE RIDFLD
deleteLockedRecord	DELETE FILE
enableStatus	INQUIRE FILE ENABLESTATUS
isAddable	INQUIRE FILE ADD
isBrowsable	INQUIRE FILE BROWSE
isDeletable	INQUIRE FILE DELETE
isEmptyOnOpen	INQUIRE FILE EMPTYSTATUS
isReadable	INQUIRE FILE READ
isRecoverable	INQUIRE FILE RECOVSTATUS
isUpdatable	INQUIRE FILE UPDATE
keyPosition	INQUIRE FILE KEYPOSITION
openStatus	INQUIRE FILE OPENSTATUS
readRecord	READ FILE
recordFormat	INQUIRE FILE RECORDFORMAT
recordLength	INQUIRE FILE RECORDSIZE
rewriteRecord	REWRITE FILE
setAccess	SET FILE ADD BROWSE DELETE etc.
setEmptyOnOpen	SET FILE EMPTYSTATUS
setStatus	SET FILE OPENSTATUS ENABLESTATUS
type	INQUIRE FILE TYPE
unlockRecord	UNLOCK FILE
writeRecord	WRITE FILE
IccFileIterator Class	

Foundation Class methods to EXEC CICS

Method	EXEC CICS
IccFileIterator (constructor)	STARTBR FILE
~IccFileIterator (destructor)	ENDBR FILE
readNextRecord	READNEXT FILE
readPreviousRecord	READPREV FILE
reset	RESETBR FILE
IccJournal Class	
Method	EXEC CICS
wait	WAIT JOURNALNUM
writeRecord	WRITE JOURNALNUM
IccProgram Class	
Method	EXEC CICS
link	LINK PROGRAM
load	LOAD PROGRAM
unload	RELEASE PROGRAM
IccResource Class	
Method	EXEC CICS
condition	(RESP & RESP2)
setRouteOption	(SYSID)
IccSemaphore Class	
Method	EXEC CICS
lock	ENQ RESOURCE
tryLock	ENQ RESOURCE NOSUSPEND
unlock	DEQ RESOURCE
IccSession Class	
Method	EXEC CICS
allocate	ALLOCATE
connectProcess	CONNECT PROCESS CONVID
converse	CONVERSE CONVID
extractProcess	EXTRACT PROCESS CONVID
flush	WAIT CONVID
free	FREE CONVID
issueAbend	ISSUE ABEND CONVID
issueConfirmation	ISSUE CONFIRMATION CONVID
issueError	ISSUE ERROR CONVID
issuePrepare	ISSUE PREPARE CONVID
issueSignal	ISSUE SIGNAL CONVID
receive	RECEIVE CONVID
send	SEND CONVID
sendInvite	SEND CONVID INVITE
sendLast	SEND CONVID LAST
state	EXTRACT ATTRIBUTES

Foundation Class methods to EXEC CICS

IccStartRequestQ Class	
Method	EXEC CICS
cancel	CANCEL
retrieveData	RETRIEVE
start	START TRANSID
IccSystem Class	
Method	EXEC CICS
applName	ASSIGN APPLID
beginBrowse	INQUIRE (FILE, TDQUEUE, etc) START
dateFormat	FORMATTIME DATEFORM
endBrowse	INQUIRE (FILE, TDQUEUE, etc) END
freeStorage	FREEMAIN
getFile	INQUIRE FILE
getNextFile	INQUIRE FILE NEXT
getStorage	GETMAIN SHARED
operatingSystem	INQUIRE SYSTEM OPSYS
operatingSystemLevel	INQUIRE SYSTEM OPREL
release	INQUIRE SYSTEM RELEASE
releaseText	INQUIRE SYSTEM RELEASE
sysId	ASSIGN SYSID
workArea	ADDRESS CWA
IccTask Class	
Method	EXEC CICS
abend	ABEND
commitUOW	SYNCPOINT
delay	DELAY
dump	DUMP TRANSACTION
enterTrace	ENTER TRACENUM
facilityType	ASSIGN STARTCODE, TERMCODE, PRINSYSID, FCI
freeStorage	FREEMAIN
isCommandSecurityOn	ASSIGN CMDSEC
isCommitSupported	ASSIGN STARTCODE
isResourceSecurityOn	ASSIGN RESSEC
isRestarted	ASSIGN RESTART
isStartDataAvailable	ASSIGN STARTCODE
principalSysId	ASSIGN PRINSYSID
priority	ASSIGN TASKPRIORITY
rollBackUOW	SYNCPOINT ROLLBACK
setPriority	CHANGE TASK PRIORITY
startType	ASSIGN STARTCODE
suspend	SUSPEND
triggerDataQueueId	ASSIGN QNAME

Foundation Class methods to EXEC CICS

userId	ASSIGN USERID
waitExternal	WAIT EXTERNAL / WAITCICS
waitOnAlarm	WAIT EVENT
workArea	ADDRESS TWA
IccTempStore Class	
Method	EXEC CICS
empty	DELETEQ TS
readItem	READQ TS ITEM
readNextItem	READQ TS NEXT
rewriteItem	WRITEQ TS ITEM REWRITE
writeItem	WRITEQ TS ITEM
IccTerminal Class	
Method	EXEC CICS
erase	SEND CONTROL ERASE
freeKeyboard	SEND CONTROL FREEKB
height	ASSIGN SCRNH
netName	ASSIGN NETNAME
receive	RECEIVE
receive3270Data	RECEIVE BUFFER
send	SEND
sendLine	SEND
setCursor	SEND CONTROL CURSOR
setLine	SEND CONTROL CURSOR
setNewLine	SEND CONTROL CURSOR
signoff	SIGNOFF
signon	SIGNON
waitForAID	RECEIVE
width	ASSIGN SCRNWD
workArea	ADDRESS TCTUA
IccTerminalData Class	
Method	EXEC CICS
alternateHeight	ASSIGN ALTSCRNH
alternateWidth	ASSIGN ALTSCRNWD
defaultHeight	ASSIGN DEFSCRNH
defaultWidth	ASSIGN DEFSCRNWD
graphicCharSetId	ASSIGN GCHARS
graphicCharCodeSet	ASSIGN GCODES
isAPLKeyboard	ASSIGN APLKYBD
isAPLText	ASSIGN APLTEXT
isBTrans	ASSIGN BTRANS
isColor	ASSIGN COLOR
isEWA	ASSIGN ESASUPP

Foundation Class methods to EXEC CICS

isExtended3270	ASSIGN EXTDS
isGoodMorning	ASSIGN GMMI
isHighlight	ASSIGN HIGHLIGHT
isKatakana	ASSIGN KATAKANA
isMSRControl	ASSIGN MSRCONTROL
isFieldOutline	ASSIGN OUTLINE
isPS	ASSIGN PS
isSOSI	ASSIGN SOSI
isTextKeyboard	ASSIGN TEXTKYBD
isTextPrint	ASSIGN TEXTPRINT
isValidation	ASSIGN VALIDATION
iccUser Class	
Method	EXEC CICS
changePassword	CHANGE PASSWORD
verifyPassword	VERIFY PASSWORD

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$buf'...
IccBuf buf1                               dal= 0 dl= 0 E+I []
IccBuf buf2(50)                            dal=50 dl= 0 E+I []
IccBuf buf3(30,fixed)                      dal=30 dl= 0 F+I []
IccBuf buf4(sizeof(AStruct),&aStruc) dal=24 dl=24 F+E [!Some text for aStruc]
IccBuf buf5(A String Literal)              dal=19 dl=19 E+I [Some data somewhere]
IccBuf buf6(buf5)                          dal=19 dl=19 E+I [Some data somewhere]
buf1 = Some XXX data for buf1              dal=22 dl=22 E+I [Some XXX data for buf1]
buf2.assign(strlen(data),data)             dal=50 dl=19 E+I [Some data somewhere]
buf1.cut(4,5)                              dal=22 dl=18 E+I [Some data for buf1]
buf5.insert(5,more,5)                      dal=24 dl=24 E+I [Some more data somewhere]
buf5.replace(4,xtra,5)                     dal=24 dl=24 E+I [Some xtra data somewhere]
buf2 << .ext                               dal=50 dl=23 E+I [Some data somewhere.ext]
buf3 = buf4                               dal=30 dl=24 F+I [!Some text for aStruc]
(buf3 == buf4) returns true (OK).
buf3 = garbage                             dal=30 dl= 7 F+I [garbage]
(buf3 != buf4) returns true (OK).
Program 'icc$buf' complete: Hit PF12 to End
```

ICC\$CLK (ICLK)

```
This is program 'icc$clk' ...
date() = [220296 ]
date(DDMMYY) = [220296 ]
date(DDMMYY,':') = [22:02:96]
date(MMDDYY) = [022296 ]
date(YYDDD) = [96053 ]
daysSince1900() = 35116
dayOfWeek() = 4                                Today is NOT Friday
dayOfMonth() = 22
monthOfYear() = 2
time() = [143832 ]
time('-') = [14-38-32]
year() = [1996]
Program 'icc$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
```

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

```

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 (IRS1)

```
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
```

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$srq1'...
Starting Tran 'ISR2' on terminal 'PE12' after 5 seconds... - <NORMAL>
request='DF!U0000'
Issuing cancel for start request='DF!U0000'...           - <NORMAL>
request='DF!U0000'
Starting Tran 'ISR2' on terminal 'PE12' after 5 seconds... - <NORMAL>
request='REQUEST1'
Program 'icc$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 rtn = [ITMP]
Start rtrm = [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 (ICCKFILE)...
>ICCKFILE< Add=true Brw=true Del=true Read=true Upd=true op=18 en=23
accessMethod=3 isRecoverable=true keyLength=3 keyPosition=16
setStatus( closed ) ... <NORMAL>
setStatus( disabled ) ... <NORMAL>
setAccess( 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
```

ICC\$TSK (ITSK)

```
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

The CICS Transaction Server for z/OS library

The published information for CICS Transaction Server for z/OS is delivered in the following forms:

The CICS Transaction Server for z/OS Information Center

The CICS Transaction Server for z/OS Information Center is the primary source of user information for CICS Transaction Server. The Information Center contains:

- Information for CICS Transaction Server in HTML format.
- Licensed and unlicensed CICS Transaction Server books provided as Adobe Portable Document Format (PDF) files. You can use these files to print hardcopy of the books. For more information, see PDF-only books.”
- Information for related products in HTML format and PDF files.

One copy of the CICS Information Center, on a CD-ROM, is provided automatically with the product. Further copies can be ordered, at no additional charge, by specifying the Information Center feature number, 7014.

Licensed documentation is available only to licensees of the product. A version of the Information Center that contains only unlicensed information is available through the publications ordering system, order number SK3T-6945.

Entitlement hardcopy books

The following essential publications, in hardcopy form, are provided automatically with the product. For more information, see The entitlement set.”

The entitlement set

The entitlement set comprises the following hardcopy books, which are provided automatically when you order CICS Transaction Server for z/OS, Version 3 Release 1:

Memo to Licensees, GI10-2559
CICS Transaction Server for z/OS Program Directory, GI10-2586
CICS Transaction Server for z/OS Release Guide, GC34-6421
CICS Transaction Server for z/OS Installation Guide, GC34-6426
CICS Transaction Server for z/OS Licensed Program Specification, GC34-6608

You can order further copies of the following books in the entitlement set, using the order number quoted above:

CICS Transaction Server for z/OS Release Guide
CICS Transaction Server for z/OS Installation Guide
CICS Transaction Server for z/OS Licensed Program Specification

PDF-only books

The following books are available in the CICS Information Center as Adobe Portable Document Format (PDF) files:

CICS books for CICS Transaction Server for z/OS

General

CICS Transaction Server for z/OS Program Directory, GI10-2586
CICS Transaction Server for z/OS Release Guide, GC34-6421
CICS Transaction Server for z/OS Migration from CICS TS Version 2.3, GC34-6425

CICS Transaction Server for z/OS Migration from CICS TS Version 1.3,
GC34-6423

CICS Transaction Server for z/OS Migration from CICS TS Version 2.2,
GC34-6424

CICS Transaction Server for z/OS Installation Guide, GC34-6426

Administration

CICS System Definition Guide, SC34-6428

CICS Customization Guide, SC34-6429

CICS Resource Definition Guide, SC34-6430

CICS Operations and Utilities Guide, SC34-6431

CICS Supplied Transactions, SC34-6432

Programming

CICS Application Programming Guide, SC34-6433

CICS Application Programming Reference, SC34-6434

CICS System Programming Reference, SC34-6435

CICS Front End Programming Interface User's Guide, SC34-6436

CICS C++ OO Class Libraries, SC34-6437

CICS Distributed Transaction Programming Guide, SC34-6438

CICS Business Transaction Services, SC34-6439

Java Applications in CICS, SC34-6440

JCICS Class Reference, SC34-6001

Diagnosis

CICS Problem Determination Guide, SC34-6441

CICS Messages and Codes, GC34-6442

CICS Diagnosis Reference, GC34-6899

CICS Data Areas, GC34-6902

CICS Trace Entries, SC34-6443

CICS Supplementary Data Areas, GC34-6905

Communication

CICS Intercommunication Guide, SC34-6448

CICS External Interfaces Guide, SC34-6449

CICS Internet Guide, SC34-6450

Special topics

CICS Recovery and Restart Guide, SC34-6451

CICS Performance Guide, SC34-6452

CICS IMS Database Control Guide, SC34-6453

CICS RACF Security Guide, SC34-6454

CICS Shared Data Tables Guide, SC34-6455

CICS DB2 Guide, SC34-6457

CICS Debugging Tools Interfaces Reference, GC34-6908

CICSplex SM books for CICS Transaction Server for z/OS

General

CICSplex SM Concepts and Planning, SC34-6459

CICSplex SM User Interface Guide, SC34-6460

CICSplex SM Web User Interface Guide, SC34-6461

Administration and Management

CICSplex SM Administration, SC34-6462

CICSplex SM Operations Views Reference, SC34-6463

CICSplex SM Monitor Views Reference, SC34-6464

CICSplex SM Managing Workloads, SC34-6465

CICSplex SM Managing Resource Usage, SC34-6466

CICSplex SM Managing Business Applications, SC34-6467

Programming

CICSplex SM Application Programming Guide, SC34-6468

CICSplex SM Application Programming Reference, SC34-6469

Diagnosis

CICSplex SM Resource Tables Reference, SC34-6470
CICSplex SM Messages and Codes, GC34-6471
CICSplex SM Problem Determination, GC34-6472

CICS family books

Communication

CICS Family: Interproduct Communication, SC34-6473
CICS Family: Communicating from CICS on System/390, SC34-6474

Licensed publications

The following licensed publications are not included in the unlicensed version of the Information Center:

CICS Diagnosis Reference, GC34-6899
CICS Data Areas, GC34-6902
CICS Supplementary Data Areas, GC34-6905
CICS Debugging Tools Interfaces Reference, GC34-6908

Other CICS books

The following publications contain further information about CICS, but are not provided as part of CICS Transaction Server for z/OS, Version 3 Release 1.

<i>Designing and Programming CICS Applications</i>	SR23-9692
<i>CICS Application Migration Aid Guide</i>	SC33-0768
<i>CICS Family: API Structure</i>	SC33-1007
<i>CICS Family: Client/Server Programming</i>	SC33-1435
<i>CICS Transaction Gateway for z/OS Administration</i>	SC34-5528
<i>CICS Family: General Information</i>	GC33-0155
<i>CICS 4.1 Sample Applications Guide</i>	SC33-1173
<i>CICS/ESA 3.3 XRF Guide</i>	SC33-0661

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.

- Ellis, Margaret A. and Bjarne Stroustrup, *The Annotated C++ Reference Manual*, Addison-Wesley Publishing Company.
- Lippman, Stanley B., *C++ Primer*, Addison-Wesley Publishing Company.
- Stroustrup, Bjarne, *The C++ Programming Language*, Addison-Wesley Publishing Company.

CICS client manuals

<i>CICS Clients: Administration</i>	SC33-1792
<i>CICS Clients: Messages</i>	SC33-1793
<i>CICS Clients: Gateways</i>	SC33-1821
<i>CICS Family: OO Programming in C++ for CICS Clients</i>	SC33-1923

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.

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully.

You can perform most tasks required to set up, run, and maintain your CICS system in one of these ways:

- using a 3270 emulator logged on to CICS
- using a 3270 emulator logged on to TSO
- using a 3270 emulator as an MVS system console

IBM Personal Communications provides 3270 emulation with accessibility features for people with disabilities. You can use this product to provide the accessibility features you need in your CICS system.

Index

Special characters

... (parameter)
in sendLine 240

Numerics

0 (zero)
in actionOnConditionAsChar 177

A

A
in actionOnConditionAsChar 177
in operatingSystem 211
abend
in IccTask class 215
in Parameter level 57
abend codes 51
abendCode
in IccAbendData class 71
abendCode (parameter)
in abend 215
abendData
in IccTask class 215
AbendDumpOpt
in Enumerations 223
in IccTask class 223
AbendHandlerOpt
in Enumerations 222
in IccTask class 222
abendTask
in ActionOnCondition 182
in CICS conditions 54
absTime
in IccClock class 99
in Type 255
absTime (parameter)
in Constructor 77
in operator= 79
access
in IccFile class 129
Access
in Enumerations 137
in IccFile class 137
access (parameter)
in setAccess 135
Accessing start data
in Starting transactions asynchronously 36
in Using CICS Services 36
accessMethod
in IccFile class 130
action (parameter)
in setActionOnAnyCondition 180
in setActionOnCondition 180
actionOnCondition
in IccResource class 177

ActionOnCondition
in Enumerations 182
in IccResource class 182
actionOnConditionAsChar
in IccResource class 177
actions (parameter)
in setActionsOnConditions 180
actionsOnConditionsText
in IccResource class 178
Activating the trace output
in Debugging Programs 50
in Tracing a Foundation Class Program 50
addable
in Access 137
address
in IccProgram class 165
AID
in IccTerminal class 233
aid (parameter)
in waitForAID 243
AIDVal
in Enumerations 244
in IccTerminal class 244
AIX, CICS for
in Platform differences 56
allocate
in IccSession class 192
AllocateOpt
in Enumerations 199
in IccSession class 199
alternateHeight
in IccTerminalData class 247
in Public methods 247
alternateWidth
in IccTerminalData class 247
in Public methods 247
append
in IccBuf class 90
applName
in IccSystem class 209
ASRAInterrupt
in IccAbendData class 71
in Public methods 71
ASRAKeyType
in IccAbendData class 72
in Public methods 72
ASRAPSW
in IccAbendData class 72
ASRARegisters
in IccAbendData class 72
in Public methods 72
ASRASpaceType
in IccAbendData class 73
in Public methods 73
ASRAStorageType
in IccAbendData class 73
in Public methods 73

- assign
 - in Example of file control 33
 - in lccBuf class 90, 91
 - in lccKey class 155
- automatic
 - in UpdateMode 103
- Automatic condition handling (callHandleEvent)
 - in CICS conditions 54
 - in Conditions, errors, and exceptions 54
- automatic creation 15
- automatic deletion 15
- auxStorage
 - in Location 228

B

- base class
 - overview 17
- Base classes
 - in Overview of the foundation classes 17
- baseName (parameter)
 - in NameOpt 87
- BASESPACE
 - in ASRASpaceType 73
- BDAM 29
- beginBrowse
 - in lccSystem class 209
- beginInsert
 - in Writing records 30
- beginInsert(VSAM only)
 - in lccFile class 130
 - in Public methods 130
- below
 - in StorageOpts 224
- blink
 - in Highlight 245
- blue
 - in Color 244
- Bool
 - in Enumerations 69
 - in lcc structure 69
- BoolSet
 - in Enumerations 69
 - in lcc structure 69
- boolText
 - in Functions 67
 - in lcc structure 67
- browsable
 - in Access 137
- browsing records 32
- Browsing records
 - in File control 32
 - in Using CICS Services 32
- buf (parameter)
 - in dump 216
 - in put 237
 - in send3270Data 239
 - in sendLine 240
 - in setData 203
- buffer
 - in Example of starting transactions 37, 38

- buffer (parameter)
 - in Constructor 90
 - in operator!= 93
 - in operator<< 93, 235
 - in operator+= 93
 - in operator= 92
 - in operator== 93
 - in Polymorphic Behavior 60
 - in put 118, 148, 179, 226
 - in registerData 202
 - in rewriteRecord 135
 - in send 238
 - in send3270Data 239
 - in sendLine 239, 240
 - in writeRecord 136
- Buffer objects
 - Data area extensibility 25
 - Data area ownership 25
 - lccBuf constructors 25
 - lccBuf methods 26
 - Working with lccResource subclasses 26
- buffers 25, 27
- byAddress
 - in LockType 189
- byValue
 - in LockType 189

C

- C++ exceptions 51
- C++ Exceptions and the Foundation Classes
 - in Conditions, errors, and exceptions 51
- callHandleEvent
 - in ActionOnCondition 182
 - in CICS conditions 54
- calling conventions 62
- Calling methods on a resource object
 - in Overview of the foundation classes 22
 - in Using CICS resources 22
- callingProgramId
 - in lccControl class 111
 - in Public methods 111
- cancel
 - in Cancelling unexpired start requests 36
 - in lccRequestId class 175
 - in lccStartRequestQ class 201
- cancelAbendHandler
 - in lccControl class 111
- cancelAlarm
 - in lccClock class 99
- Cancelling unexpired start requests
 - in Starting transactions asynchronously 36
 - in Using CICS Services 36
- Case
 - in Enumerations 244
 - in lccTerminal class 244
- caseOpt (parameter)
 - in receive 237
 - in receive3270Data 237

- catch
 - in C++ Exceptions and the Foundation Classes 51, 52
 - in Exception handling (throwException) 55, 56
 - in main function 276
- catchException
 - in Functions 67
 - in lcc structure 67
- CEDF (CICS Execution Diagnostic Facility) 50
- ch (parameter)
 - in operator<< 94, 235
- changePassword
 - in lccUser class 265
 - in Public methods 265
- char*
 - in C++ Exceptions and the Foundation Classes 52
- CheckOpt
 - in Enumerations 206
 - in lccStartRequestQ class 206
- CICS
 - in ASRAStorageType 73
 - in GetOpt 70
- CICS conditions
 - abendTask 56
 - automatic condition handling 54
 - Automatic condition handling (callHandleEvent) 54
 - callHandleEvent 54
 - exception handling 55
 - Exception handling (throwException) 55
 - in Conditions, errors, and exceptions 53
 - manual condition handling 54
 - Manual condition handling (noAction) 54
 - noAction 54
 - severe error handling 56
 - Severe error handling (abendTask) 56
 - throwException 55
- CICS Execution Diagnostic Facility (CEDF) 50
- CICS for AIX
 - in Platform differences 56
- CICS OS/2
 - in Platform differences 56
- CICS resources 21
- CICSCondition
 - in C++ Exceptions and the Foundation Classes 53
 - in Type 127
- CICSDataKey
 - in StorageOpts 224
- CICSEXECKEY
 - in ASRAKeyType 72
- CICSInternalTask
 - in StartType 223
- CICSTS13.CICS.SDFHSAMP 6
- CICSTS31.CICS.SDFHC370 6
- CICSTS31.CICS.SDFHLOAD 7
- CICSTS31.CICS.SDFHPROC 6
- CICSTS31.CICS.SDFHSAMP 6
- CICSTS31.CICS.SDFHSDCK 6
- class
 - base 17
 - resource 19
 - resource identification 18
- class (*continued*)
 - singleton 22
 - support 20
- ClassMemoryMgmt
 - in Enumerations 69
 - in lcc structure 69
- className
 - in lccBase class 85
 - in lccEvent class 123
 - in lccException class 126
 - in lccMessage class 161
- className (parameter)
 - in Constructor 125, 161
 - in setClassName 86
- classType
 - in lccBase class 85
 - in lccEvent class 123
 - in lccException class 126
- ClassType
 - in Enumerations 87
 - in lccBase class 87
- classType (parameter)
 - in Constructor 125, 177
- clear
 - in Example of polymorphic behavior 61
 - in lccDataQueue class 117
 - in lccResource class 178
 - in lccTempStore class 225
 - in lccTerminal class 233
 - in Polymorphic Behavior 60
- CLEAR
 - in AIDVal 244
- clearData
 - in lccStartRequestQ class 201
- clearInputMessage
 - in lccProgram class 165
- clearPrefix
 - in lccJournal class 147
- closed
 - in Status 138
- cmmCICS
 - in ClassMemoryMgmt 70
 - in Storage management 62
- cmmDefault
 - in ClassMemoryMgmt 69
 - in Storage management 62
- cmmNonCICS
 - in ClassMemoryMgmt 70
 - in Storage management 62
- CODE/370 50
- Codes
 - in Enumerations 105
 - in lccCondition structure 105
- col (parameter)
 - in send 238
 - in send3270Data 239
 - in sendLine 240
 - in setCursor 241
- Color
 - in Enumerations 244
 - in lccTerminal class 244

- color (parameter)
 - in operator<< 235
 - in setColor 240
- commArea
 - in lccControl class 111
- commArea (parameter)
 - in link 166
 - in setNextCommArea 241
- commitOnReturn
 - in CommitOpt 168
- CommitOpt
 - in Enumerations 168
 - in lccProgram class 168
- commitUOW
 - in lccTask class 215
- Compile and link "Hello World"
 - in Hello World 10
- compiling programs 49
- Compiling Programs
 - in Compiling, executing, and debugging 49
- Compiling, executing, and debugging
 - Execution Diagnostic Facility 50
 - Symbolic Debuggers 50
 - Tracing a Foundation Class Program 50
- complete
 - in Kind 157
- complete key 30
- completeLength
 - in lccKey class 155
 - in Public methods 155
- completeLength (parameter)
 - in Constructor 155
- condition
 - in lccEvent class 123
 - in lccResource class 178
 - in Manual condition handling (noAction) 54
 - in Resource classes 19
- condition (parameter)
 - in actionOnCondition 177
 - in actionOnConditionAsChar 177
 - in conditionText 67
 - in setActionOnCondition 180
- condition 0 (NORMAL)
 - in actionsOnConditionsText 178
- condition 1 (ERROR)
 - in actionsOnConditionsText 178
- condition 2 (RDATT)
 - in actionsOnConditionsText 178
- condition 3 (WRBRK)
 - in actionsOnConditionsText 178
- condition 4 (ICCEOF)
 - in actionsOnConditionsText 178
- condition 5 (EODS)
 - in actionsOnConditionsText 178
- condition 6 (EOC)
 - in actionsOnConditionsText 178
- Conditions, errors, and exceptions
 - Automatic condition handling (callHandleEvent) 54
 - Exception handling (throwException) 55
 - Manual condition handling (noAction) 54
 - Method level 57
- Conditions, errors, and exceptions *(continued)*
 - Object level 56
 - Parameter level 57
 - Severe error handling (abendTask) 56
- conditionText
 - in Functions 67
 - in lcc structure 67
 - in lccEvent class 124
 - in lccResource class 178
- ConditionType
 - in Enumerations 182
 - in lccResource class 182
- confirmation
 - in SendOpt 199
- connectProcess
 - in lccSession class 192
 - in Public methods 192
- console
 - in lccControl class 112
- const
 - in Glossary 299
- Constructor
 - in lccAbendData class 71
 - in lccAbendData constructor (protected) 71
 - in lccAbsTime class 77
 - in lccAbsTime constructor 77
 - in lccAlarmRequestId class 81
 - in lccAlarmRequestId constructors 81
 - in lccBase class 85
 - in lccBase constructor (protected) 85
 - in lccBuf class 89, 90
 - in lccBuf constructors 89, 90
 - in lccClock class 99
 - in lccClock constructor 99
 - in lccConsole class 107
 - in lccConsole constructor (protected) 107
 - in lccControl class 111
 - in lccControl constructor (protected) 111
 - in lccConvId class 115
 - in lccConvId constructors 115
 - in lccDataQueue class 117
 - in lccDataQueue constructors 117
 - in lccDataQueueId class 121
 - in lccDataQueueId constructors 121
 - in lccEvent class 123
 - in lccEvent constructor 123
 - in lccException class 125
 - in lccException constructor 125
 - in lccFile class 129
 - in lccFile constructors 129
 - in lccFileId class 139
 - in lccFileId constructors 139
 - in lccFileIterator class 141
 - in lccFileIterator constructor 141
 - in lccGroupId class 145
 - in lccGroupId constructors 145
 - in lccJournal class 147
 - in lccJournal constructors 147
 - in lccJournalId class 151
 - in lccJournalId constructors 151
 - in lccJournalTypeId class 153

Constructor (*continued*)
 in lccJournalTypeld constructors 153
 in lccKey class 155
 in lccKey constructors 155
 in lccLockId class 159
 in lccLockId constructors 159
 in lccMessage class 161
 in lccMessage constructor 161
 in lccPartnerId class 163
 in lccPartnerId constructors 163
 in lccProgram class 165
 in lccProgram constructors 165
 in lccProgramId class 169
 in lccProgramId constructors 169
 in lccRBA class 171
 in lccRBA constructor 171
 in lccRecordIndex class 173
 in lccRecordIndex constructor (protected) 173
 in lccRequestId class 175
 in lccRequestId constructors 175
 in lccResource class 177
 in lccResource constructor (protected) 177
 in lccResourceId class 183
 in lccResourceId constructors (protected) 183
 in lccRRN class 185
 in lccRRN constructors 185
 in lccSemaphore class 187
 in lccSemaphore constructor 187
 in lccSession class 191
 in lccSession constructor (protected) 191
 in lccSession constructors (public) 191
 in lccStartRequestQ class 201
 in lccStartRequestQ constructor (protected) 201
 in lccSysId class 207
 in lccSysId constructors 207
 in lccSystem class 209
 in lccSystem constructor (protected) 209
 in lccTask class 215
 in lccTask Constructor (protected) 215
 in lccTempStore class 225
 in lccTempStore constructors 225
 in lccTempStoreId class 229
 in lccTempStoreId constructors 229
 in lccTermId class 231
 in lccTermId constructors 231
 in lccTerminal class 233
 in lccTerminal constructor (protected) 233
 in lccTerminalData class 247
 in lccTerminalData constructor (protected) 247
 in lccTime class 253
 in lccTime constructor (protected) 253
 in lccTimeInterval class 257
 in lccTimeInterval constructors 257
 in lccTimeOfDay class 259
 in lccTimeOfDay constructors 259
 in lccTPNameId class 261
 in lccTPNameId constructors 261
 in lccTransId class 263
 in lccTransId constructors 263
 in lccUser class 265
 in lccUser constructors 265

Constructor (*continued*)
 in lccUserId class 269
 in lccUserId constructors 269
 converse
 in lccSession class 193
 convId
 in lccSession class 193
 convId (parameter)
 in Constructor 115
 convName (parameter)
 in Constructor 115
 in operator= 115
 copt (parameter)
 in setStartOpts 204
 createDump
 in AbendDumpOpt 223
 creating a resource object 21
 Creating a resource object
 in Overview of the foundation classes 21
 in Using CICS resources 21
 Singleton classes 22
 Creating an object
 in C++ Objects 15
 creating object 15
 current (parameter)
 in setPrefix 148
 cursor
 in Finding out information about a terminal 44
 in lccTerminal class 233
 customClassNum
 in lccBase class 86
 in Public methods 86
 cut
 in lccBuf class 91
 in lccBuf constructors 26
 CVDA
 in Enumeration 271
 in lccValue structure 271
 cyan
 in Color 244

D

data
 in Accessing start data 36
 in Finding out information about a terminal 44
 in lccStartRequestQ class 202
 in lccTerminal class 233
 in lccTerminalData class 247
 data (parameter)
 in enterTrace 217
 in put 196
 data area extensibility 25
 Data area extensibility
 in Buffer objects 25
 in lccBuf class 25
 data area ownership 25
 Data area ownership
 in Buffer objects 25
 in lccBuf class 25

- dataArea
 - in lccBuf class 91
- dataArea (parameter)
 - in append 90
 - in assign 90, 155
 - in Constructor 89
 - in insert 92
 - in overlay 95
 - in replace 96
- dataAreaLength
 - in lccBuf class 91
 - in Public methods 91
- dataAreaOwner
 - in Data area ownership 25
 - in lccBuf class 91
- DataAreaOwner
 - in Enumerations 97
 - in lccBuf class 97
- dataAreaType
 - in Data area extensibility 25
 - in lccBuf class 91
- DataAreaType
 - in Enumerations 97
 - in lccBuf class 97
- dataItems
 - in Example of polymorphic behavior 60
- dataLength
 - in lccBuf class 92
- dataqueue
 - in FacilityType 223
- dataQueueTrigger
 - in StartType 223
- date
 - in lccAbsTime class 77
 - in lccClock class 99
- date services 45
- dateFormat
 - in lccSystem class 210
- DateFormat
 - in Enumerations 102
 - in lccClock class 102
- dateSeparator (parameter)
 - in date 77, 100
 - in Example of time and date services 46
- dayOfMonth
 - in Example of time and date services 46
 - in lccAbsTime class 77
 - in lccClock class 100
- dayOfWeek
 - in Example of time and date services 46
 - in lccAbsTime class 78
 - in lccClock class 100
- DayOfWeek
 - in Enumerations 102
 - in lccClock class 102
- daysSince1900
 - in Example of time and date services 46
 - in lccAbsTime class 78
 - in lccClock class 100
- daysUntilPasswordExpires
 - in lccUser class 266
- dComplete
 - in DumpOpts 223
- dDCT
 - in DumpOpts 223
- dDefault
 - in DumpOpts 223
- debuggers 50
- debugging programs 49
- Debugging Programs
 - Activating the trace output 50
 - Enabling EDF 50
 - Execution Diagnostic Facility 50
 - in Compiling, executing, and debugging 49
 - Symbolic Debuggers 50
 - Tracing a Foundation Class Program 50
- defaultColor
 - in Color 244
- defaultHeight
 - in lccTerminalData class 248
 - in Public methods 248
- defaultHighlight
 - in Highlight 245
- defaultWidth
 - in lccTerminalData class 248
 - in Public methods 248
- delay
 - in lccTask class 216
 - in Support Classes 21
- deletable
 - in Access 137
- delete
 - in Deleting an object 16
 - in Storage management 62
- delete operator 15
- deleteLockedRecord 32
 - in Deleting locked records 32
 - in lccFile class 130
- deleteRecord
 - in Deleting normal records 31
 - in lccFile class 130
- deleteRecord method 31
- Deleting an object
 - in C++ Objects 16
- deleting items 42
- Deleting items
 - in Temporary storage 42
 - in Using CICS Services 42
- Deleting locked records
 - in Deleting records 32
 - in File control 32
- Deleting normal records
 - in Deleting records 31
 - in File control 31
- deleting queues 40
- Deleting queues
 - in Transient Data 40
 - in Using CICS Services 40
- deleting records 31
- Deleting records
 - Deleting locked records 32
 - Deleting normal records 31

- Deleting records (*continued*)
 - in File control 31
 - in Using CICS Services 31
- dFCT
 - in DumpOpts 223
- DFHCURDI 7
- DFHCURDS 6, 7
- disabled
 - in Status 138
- doSomething
 - in Using an object 16
- dPCT
 - in DumpOpts 223
- DPL
 - in StartType 223
- dPPT
 - in DumpOpts 223
- dProgram
 - in DumpOpts 223
- dSIT
 - in DumpOpts 223
- dStorage
 - in DumpOpts 223
- dTables
 - in DumpOpts 223
- dTask
 - in DumpOpts 223
- dTCT
 - in DumpOpts 223
- dTerminal
 - in DumpOpts 223
- dTRT
 - in DumpOpts 223
- dump
 - in IccTask class 216
- dumpCode (parameter)
 - in dump 216
- DumpOpts
 - in Enumerations 223
 - in IccTask class 223
- dynamic creation 15
- dynamic deletion 15
- dynamic link library 6
- Dynamic link library
 - in Installed contents 6
 - Location 6

E

- ECBList (parameter)
 - in waitExternal 221
- EDF (Execution Diagnostic Facility) 50
- EDF (parameter)
 - in initializeEnvironment 68
- empty
 - in Deleting items 42
 - in Deleting queues 40
 - in IccDataQueue class 117
 - in IccTempStore class 226
 - in Temporary storage 41
 - in Transient Data 39

- enabled
 - in Status 138
- enableStatus
 - in IccFile class 131
- Enabling EDF
 - in Debugging Programs 50
 - in Execution Diagnostic Facility 50
- endBrowse
 - in IccSystem class 210
- endInsert
 - in Writing records 30
- endInsert(VSAM only)
 - in IccFile class 131
 - in Public methods 131
- endl
 - in Example of terminal control 45
- ENTER
 - in AIDVal 244
- enterTrace
 - in IccTask class 217
- entryPoint
 - in IccProgram class 166
- Enumeration
 - CVDA 271
 - in IccValue structure 271
- Enumerations
 - AbendDumpOpt 223
 - AbendHandlerOpt 222
 - Access 137
 - ActionOnCondition 182
 - AIDVal 244
 - AllocateOpt 199
 - Bool 69
 - BoolSet 69
 - Case 244
 - CheckOpt 206
 - ClassMemoryMgmt 69
 - ClassType 87
 - Codes 105
 - Color 244
 - CommitOpt 168
 - ConditionType 182
 - DataAreaOwner 97
 - DataAreaType 97
 - DateFormat 102
 - DayOfWeek 102
 - DumpOpts 223
 - FacilityType 223
 - FamilySubset 70
 - GetOpt 70
 - HandleEventReturnOpt 182
 - Highlight 245
 - in Icc structure 69
 - in IccBase class 87
 - in IccBuf class 97
 - in IccClock class 102
 - in IccCondition structure 105
 - in IccConsole class 110
 - in IccException class 127
 - in IccFile class 137
 - in IccJournal class 150

- Enumerations (*continued*)
 - in `IccKey` class 157
 - in `IccProgram` class 168
 - in `IccRecordIndex` class 174
 - in `IccResource` class 182
 - in `IccSemaphore` class 189
 - in `IccSession` class 199
 - in `IccStartRequestQ` class 206
 - in `IccSystem` class 213
 - in `IccTask` class 222
 - in `IccTempStore` class 228
 - in `IccTerminal` class 244
 - in `IccTime` class 255
 - Kind 157
 - LifeTime 189
 - LoadOpt 168
 - Location 228
 - LockType 189
 - MonthOfYear 102
 - NameOpt 87
 - NextTransIdOpt 245
 - NoSpaceOpt 228
 - Options 150
 - Platforms 70
 - ProtectOpt 206
 - Range 105
 - ReadMode 137
 - ResourceType 213
 - RetrieveOpt 206
 - SearchCriterion 138
 - SendOpt 199
 - SeverityOpt 110
 - StartType 223
 - StateOpt 199
 - Status 138
 - StorageOpts 224
 - SyncLevel 200
 - TraceOpt 224
 - Type 127, 174, 255
 - UpdateMode 103
 - WaitPostType 224
 - WaitPurgeability 224
- `equalToKey`
 - in `SearchCriterion` 138
- `erase`
 - in Example of terminal control 45
 - in Hello World 9
 - in `IccTerminal` class 234
 - in Sending data to a terminal 43
- `errorCode`
 - in `IccSession` class 193
- ESDS
 - in File control 29
- ESDS file 29
- ESMReason
 - in `IccUser` class 266
- ESMResponse
 - in `IccUser` class 266
- event (parameter)
 - in `handleEvent` 179
- Example of file control
 - in File control 32
 - in Using CICS Services 32
- Example of managing transient data
 - in Transient Data 40
 - in Using CICS Services 40
- Example of polymorphic behavior
 - in Miscellaneous 60
 - in Polymorphic Behavior 60
- Example of starting transactions
 - in Starting transactions asynchronously 36
 - in Using CICS Services 36
- Example of Temporary Storage
 - in Temporary storage 42
 - in Using CICS Services 42
- Example of terminal control
 - in Terminal control 44
 - in Using CICS Services 44
- Example of time and date services
 - in Time and date services 45
 - in Using CICS Services 45
- exception
 - in `TraceOpt` 224
- exception (parameter)
 - in `catchException` 67
- Exception handling (`throwException`)
 - in CICS conditions 55
 - in Conditions, errors, and exceptions 55
- `exceptionNum` (parameter)
 - in Constructor 125
- exceptions 51
- `exceptionType` (parameter)
 - in Constructor 125
- Executing Programs
 - in Compiling, executing, and debugging 49
- Execution Diagnostic Facility
 - Enabling EDF 50
 - in Compiling, executing, and debugging 50
 - in Debugging Programs 50
- Execution Diagnostic Facility (EDF) 50
- Expected Output from "Hello World"
 - in Hello World 10
 - in Running "Hello World" on your CICS server 10
- extensible
 - in `DataAreaType` 97
- external
 - in `DataAreaOwner` 97
- `extractProcess`
 - in `IccSession` class 193
- `extractState`
 - in `StateOpt` 200

F

- `facilityType`
 - in `IccTask` class 217
- FacilityType
 - in Enumerations 223
 - in `IccTask` class 223
- fam (parameter)
 - in `initializeEnvironment` 68

- familyConformanceError
 - in C++ Exceptions and the Foundation Classes 53
 - in Type 128
- FamilySubset
 - in Enumerations 70
 - in lcc structure 70
- FEPIRequest
 - in StartType 223
- file (parameter)
 - in Constructor 141
 - in Example of file control 33
- file control
 - browsing records 32
 - deleting records 31
 - example 32
 - rewriting records 31
 - updating records 31
- File control
 - Browsing records 32
 - Deleting locked records 32
 - Deleting normal records 31
 - Deleting records 31
 - Example of file control 32
 - in Using CICS Services 29
 - Reading ESDS records 30
 - Reading KSDS records 30
 - Reading records 29
 - Reading RRDS records 30
 - Updating records 31
 - Writing ESDS records 31
 - Writing KSDS records 31
 - Writing records 30
 - Writing RRDS records 31
- fileName (parameter)
 - in Constructor 129, 139
 - in getFile 210
 - in operator= 139
- Finding out information about a terminal
 - in Terminal control 44
 - in Using CICS Services 44
- First Screen
 - in ICC\$PRG1 (IPR1) 294
 - in Output from sample programs 294
- fixed
 - in DataAreaType 97
- flush
 - in Example of terminal control 45
 - in lccSession class 194
- for
 - in Example of file control 33
- Form
 - in Polymorphic Behavior 60
- format (parameter)
 - in append 90
 - in assign 91
 - in date 77, 100
 - in Example of time and date services 46
 - in send 238
 - in send3270Data 239
 - in sendLine 240
- Foundation Class Abend codes
 - in Conditions, errors, and exceptions 51
- free
 - in lccSession class 194
- freeKeyboard
 - in lccTerminal class 234
 - in Sending data to a terminal 43
- freeStorage
 - in lccSystem class 210
 - in lccTask class 217
- fsAllowPlatformVariance
 - in FamilySubset 70
 - in Platform differences 56
- fsDefault
 - in FamilySubset 70
- fsEnforce
 - in FamilySubset 70
 - in Platform differences 56
- fullAccess
 - in Access 137
- Functions
 - boolText 67
 - catchException 67
 - conditionText 67
 - in lcc structure 67
 - initializeEnvironment 67
 - isClassMemoryMgmtOn 68
 - isEDFOn 68
 - isFamilySubsetEnforcementOn 68
 - returnToCICS 68
 - setEDF 68
 - unknownException 69

G

- generic
 - in Kind 157
- generic key 30
- get
 - in Example of polymorphic behavior 61
 - in lccDataQueue class 117
 - in lccResource class 178
 - in lccSession class 194
 - in lccTempStore class 226
 - in lccTerminal class 234
 - in Polymorphic Behavior 60
- getFile
 - in lccSystem class 210
- getNextFile
 - in lccSystem class 211
- GetOpt
 - in Enumerations 70
 - in lcc structure 70
- getStorage
 - in lccSystem class 211
 - in lccTask class 217
- gid (parameter)
 - in Constructor 265
- graphicCharCodeSet
 - in lccTerminalData class 248

- graphicCharSetId
 - in IccTerminalData class 248
- green
 - in Color 244
- groupId
 - in IccUser class 266
- groupName (parameter)
 - in Constructor 145, 265
 - in operator= 145
- gteqToKey
 - in SearchCriterion 138

H

- H
 - in actionOnConditionAsChar 177
- handleEvent
 - in Automatic condition handling (callHandleEvent) 55
 - in IccResource class 179
- HandleEventReturnOpt
 - in Enumerations 182
 - in IccResource class 182
- handPost
 - in WaitPostType 224
- Header files
 - in Installed contents 5
 - Location 6
- height
 - in IccTerminal class 234
- Hello World
 - commentary 9
 - Compile and link 10
 - Expected Output from "Hello World" 10
 - running 10
- Highlight
 - in Enumerations 245
 - in IccTerminal class 245
- highlight (parameter)
 - in operator<< 235
 - in setHighlight 241
- hold
 - in LoadOpt 168
- hours
 - in IccAbsTime class 78
 - in IccTime class 253
- hours (parameter)
 - in Constructor 253, 257, 259
 - in set 257, 259

I

- Icc
 - in Foundation Classes—reference 66
 - in Method level 57
 - in Overview of the foundation classes 17
- Icc structure
 - Bool 69
 - BoolSet 69
 - boolText 67
 - catchException 67

- Icc structure (*continued*)
 - ClassMemoryMgmt 69
 - conditionText 67
 - FamilySubset 70
 - GetOpt 70
 - initializeEnvironment 67
 - isClassMemoryMgmtOn 68
 - isEDFOn 68
 - isFamilySubsetEnforcementOn 68
 - Platforms 70
 - returnToCICS 68
 - setEDF 68
 - unknownException 69
- Icc::initializeEnvironment
 - in Storage management 62
- ICC\$BUF 6
- ICC\$BUF (IBUF)
 - in Output from sample programs 291
- ICC\$CLK 6
- ICC\$CLK (ICLK)
 - in Output from sample programs 291
- ICC\$DAT (IDAT)
 - in Output from sample programs 291
- ICC\$EXC1 (IEX1)
 - in Output from sample programs 292
- ICC\$EXC2 (IEX2)
 - in Output from sample programs 292
- ICC\$EXC3 (IEX3)
 - in Output from sample programs 292
- ICC\$FIL (IFIL)
 - in Output from sample programs 293
- ICC\$HEL 6
- ICC\$HEL (IHEL)
 - in Output from sample programs 293
- ICC\$JRN (IJRN)
 - in Output from sample programs 293
- ICC\$PRG1 (IPR1)
 - First Screen 294
 - in Output from sample programs 294
 - Second Screen 294
- ICC\$RES1 (IRS1)
 - in Output from sample programs 294
- ICC\$RES2 (IRS2)
 - in Output from sample programs 295
- ICC\$SEM (ISEM)
 - in Output from sample programs 295
- ICC\$SES1 6
- ICC\$SES1 (ISE1)
 - in Output from sample programs 295
- ICC\$SES2 6
 - in Output from sample programs 296
- ICC\$SRQ1 (ISR1)
 - in Output from sample programs 296
- ICC\$SRQ2 (ISR2)
 - in Output from sample programs 296
- ICC\$SYS (ISYS)
 - in Output from sample programs 297
- ICC\$TMP (ITMP)
 - in Output from sample programs 297
- ICC\$TRM (ITRM)
 - in Output from sample programs 298

- ICC\$TSK (ITSK)
 - in Output from sample programs 298
- lccAbendData
 - in Singleton classes 22
- lccAbendData class
 - abendCode 71
 - ASRAInterrupt 71
 - ASRAKeyType 72
 - ASRAPSW 72
 - ASRARegisters 72
 - ASRASpaceType 73
 - ASRAStorageType 73
 - Constructor 71
 - instance 74
 - isDumpAvailable 74
 - originalAbendCode 74
 - programName 74
- lccAbendData constructor (protected)
 - Constructor 71
 - in lccAbendData class 71
- lccAbsTime
 - in Base classes 18
 - in delay 216
 - in lccTime class 253
 - in Support Classes 21
 - in Time and date services 45
- lccAbsTime class
 - Constructor 77
 - date 77
 - dayOfMonth 77
 - dayOfWeek 78
 - daysSince1900 78
 - hours 78
 - milliSeconds 78
 - minutes 78
 - monthOfYear 78
 - operator= 79
 - packedDecimal 79
 - seconds 79
 - time 79
 - timeInHours 79
 - timeInMinutes 79
 - timeInSeconds 79
 - year 79
- lccAbsTime constructor
 - Constructor 77
 - in lccAbsTime class 77
- lccAbsTime,
 - in Support Classes 21
- lccAlarmRequestId
 - in lccAlarmRequestId class 81
- lccAlarmRequestId class
 - Constructor 81
 - isExpired 81
 - operator= 82
 - setTimerECA 82
 - timerECA 82
- lccAlarmRequestId constructors
 - Constructor 81
 - in lccAlarmRequestId class 81
- lccBase
 - in Base classes 17
 - in Foundation Classes—reference 66
 - in lccAbendData class 71
 - in lccAbsTime class 77
 - in lccAlarmRequestId class 81
 - in lccBase class 85
 - in lccBuf class 89
 - in lccClock class 99
 - in lccConsole class 107
 - in lccControl class 111
 - in lccConvId class 115
 - in lccDataQueue class 117
 - in lccDataQueueId class 121
 - in lccEvent class 123
 - in lccException class 125
 - in lccFile class 129
 - in lccFileId class 139
 - in lccFileIterator class 141
 - in lccGroupId class 145
 - in lccJournal class 147
 - in lccJournalId class 151
 - in lccJournalTypeId class 153
 - in lccKey class 155
 - in lccLockId class 159
 - in lccMessage class 161
 - in lccPartnerId class 163
 - in lccProgram class 165
 - in lccProgramId class 169
 - in lccRBA class 171
 - in lccRecordIndex class 173
 - in lccRequestId class 175
 - in lccResource class 177
 - in lccResourceId class 183
 - in lccRRN class 185
 - in lccSemaphore class 187
 - in lccSession class 191
 - in lccStartRequestQ class 201
 - in lccSysId class 207
 - in lccSystem class 209
 - in lccTask class 215
 - in lccTempStore class 225
 - in lccTempStoreId class 229
 - in lccTermId class 231
 - in lccTerminal class 233
 - in lccTerminalData class 247
 - in lccTime class 253
 - in lccTimeInterval class 257
 - in lccTimeOfDay class 259
 - in lccTPNameId class 261
 - in lccTransId class 263
 - in lccUser class 265
 - in lccUserId class 269
 - in Resource classes 19
 - in Resource identification classes 18
 - in Storage management 62
 - in Support Classes 20
- lccBase class
 - className 85
 - classType 85
 - ClassType 87

- lccBase class *(continued)*
 - Constructor 85
 - customClassNum 86
 - NameOpt 87
 - operator delete 86
 - operator new 86
 - overview 17
 - setClassName 86
 - setCustomClassNum 86
- lccBase constructor (protected)
 - Constructor 85
 - in lccBase class 85
- lccBuf
 - in Buffer objects 25
 - in C++ Exceptions and the Foundation Classes 53
 - in Data area extensibility 25
 - in Data area ownership 25
 - in Example of file control 33
 - in Example of managing transient data 40
 - in Example of polymorphic behavior 61
 - in Example of starting transactions 37, 38, 39
 - in Example of Temporary Storage 42, 43
 - in Example of terminal control 44
 - in lccBuf class 25, 89
 - in lccBuf constructors 25, 26
 - in lccBuf methods 26
 - in Reading data 39
 - in Reading items 41
 - in Scope of data in lccBuf reference returned from 'read' methods 63
 - in Support Classes 21
 - in Working with lccResource subclasses 27
- lccBuf class
 - append 90
 - assign 90, 91
 - Constructor 89, 90
 - constructors 25
 - cut 91
 - data area extensibility 25
 - Data area extensibility 25
 - data area ownership 25
 - Data area ownership 25
 - dataArea 91
 - dataAreaLength 91
 - dataAreaOwner 91
 - DataAreaOwner 97
 - dataAreaType 91
 - DataAreaType 97
 - dataLength 92
 - lccBuf constructors 25
 - lccBuf methods 26
 - in Buffer objects 25
 - insert 92
 - isFMHContained 92
 - methods 26
 - operator const char* 92
 - operator!= 93
 - operator<< 93, 95
 - operator+= 93
 - operator= 92
 - operator== 93
- lccBuf class *(continued)*
 - overlay 95
 - replace 95
 - setDataLength 96
 - setFMHContained 96
 - Working with lccResource subclasses 26
- lccBuf constructors 25
 - Constructor 89, 90
 - in Buffer objects 25
 - in lccBuf class 25, 89
- lccBuf methods 26
 - in Buffer objects 26
 - in lccBuf class 26
- lccBuf reference 63
- lccClock
 - in Example of time and date services 45, 46
 - in lccAlarmRequestId class 81
 - in lccClock class 99
 - in Time and date services 45
- lccClock class
 - absTime 99
 - cancelAlarm 99
 - Constructor 99
 - date 99
 - DateFormat 102
 - dayOfMonth 100
 - dayOfWeek 100
 - DayOfWeek 102
 - daysSince1900 100
 - milliseconds 100
 - monthOfYear 100
 - MonthOfYear 102
 - setAlarm 101
 - time 101
 - update 101
 - UpdateMode 103
 - year 101
- lccClock constructor
 - Constructor 99
 - in lccClock class 99
- lccCondition
 - in C++ Exceptions and the Foundation Classes 53
- lccCondition structure
 - Codes 105
 - Range 105
- lccConsole
 - in Buffer objects 25
 - in Object level 57
 - in Singleton classes 22
- lccConsole class
 - Constructor 107
 - instance 107
 - overview 22
 - put 107
 - replyTimeout 107
 - resetRouteCodes 108
 - setAllRouteCodes 108
 - setReplyTimeout 108
 - setRouteCodes 108
 - SeverityOpt 110
 - write 108

- lccConsole class (*continued*)
 - writeAndGetReply 109
- lccConsole constructor (protected)
 - Constructor 107
 - in lccConsole class 107
- lccControl
 - in Base classes 17
 - in Example of starting transactions 37, 38
 - in Hello World 9
 - in lccControl class 111
 - in lccProgram class 165
 - in main function 275, 276
 - in Mapping EXEC CICS calls to Foundation Class methods 285
 - in Method level 57
 - in Singleton classes 22
 - in Support Classes 21
- lccControl class
 - callingProgramId 111
 - cancelAbendHandler 111
 - commArea 111
 - console 112
 - Constructor 111
 - initData 112
 - instance 112
 - isCreated 112
 - overview 17, 22
 - programId 112
 - resetAbendHandler 112
 - returnProgramId 113
 - run 113
 - session 113
 - setAbendHandler 113
 - startRequestQ 113
 - system 113
 - task 114
 - terminal 114
- lccControl constructor (protected)
 - Constructor 111
 - in lccControl class 111
- lccControl::run
 - in Mapping EXEC CICS calls to Foundation Class methods 285
- lccConvId
 - in lccConvId class 115
- lccConvId class
 - Constructor 115
 - operator= 115
- lccConvId constructors
 - Constructor 115
 - in lccConvId class 115
- lccDataQueue
 - in Buffer objects 25
 - in Example of managing transient data 40
 - in Example of polymorphic behavior 61
 - in Resource classes 19
 - in Temporary storage 41
 - in Transient Data 39
 - in Working with lccResource subclasses 27
 - in Writing data 40
- lccDataQueue class
 - clear 117
 - Constructor 117
 - empty 117
 - get 117
 - put 118
 - readItem 118
 - writeItem 118
- lccDataQueue constructors
 - Constructor 117
 - in lccDataQueue class 117
- lccDataQueueId
 - in Example of managing transient data 40
 - in lccDataQueueId class 121
 - in Transient Data 39
- lccDataQueueId class
 - Constructor 121
 - operator= 121
- lccDataQueueId constructors
 - Constructor 121
 - in lccDataQueueId class 121
- lccEvent
 - in lccEvent class 123
 - in Support Classes 21
- lccEvent class
 - className 123
 - classType 123
 - condition 123
 - conditionText 124
 - Constructor 123
 - methodName 124
 - summary 124
- lccEvent constructor
 - Constructor 123
 - in lccEvent class 123
- lccException
 - in C++ Exceptions and the Foundation Classes 52, 53
 - in lccException class 125
 - in lccMessage class 161
 - in main function 276
 - in Method level 57
 - in Object level 57
 - in Parameter level 58
 - in Support Classes 21
- lccException class
 - CICSCondition type 53
 - className 126
 - classType 126
 - Constructor 125
 - familyConformanceError type 53
 - internalError type 53
 - invalidArgument type 53
 - invalidMethodCall type 53
 - message 126
 - methodName 126
 - number 126
 - objectCreationError type 52
 - summary 126
 - type 127
 - Type 127

lccException class *(continued)*

typeText 127

lccException constructor

Constructor 125

in lccException class 125

ICCFCC 6

ICCFCL 6

ICCFCDLL 6

ICCFCGL 6

ICCFCIMP 6

ICCFCL 7

lccFile

in Browsing records 32

in Buffer objects 25

in C++ Exceptions and the Foundation Classes 53

in Deleting locked records 32

in Deleting normal records 31

in Example of file control 32

in File control 29

in lccFile class 129

in lccFileIterator class 141

in Reading ESDS records 30

in Reading KSDS records 30

in Reading records 29, 30

in Reading RRDS records 30

in Resource identification classes 18

in Singleton classes 22

in Updating records 31

in Writing ESDS records 31

in Writing KSDS records 31

in Writing records 30

in Writing RRDS records 31

lccFile class

access 129

Access 137

accessMethod 130

beginInsert(VSAM only) 130

Constructor 129

deleteLockedRecord 32, 130

deleteRecord 130

deleteRecord method 31

enableStatus 131

endInsert(VSAM only) 131

isAddable 131

isBrowsable 131

isDeletable 131

isEmptyOnOpen 132

isReadable 132

isReadable method 30

isRecoverable 132

isUpdatable 132

keyLength 132

keyLength method 30

keyPosition 133

keyPosition method 30

openStatus 133

ReadMode 137

readRecord 133

readRecord method 30

recordFormat 134

recordFormat method 30

lccFile class *(continued)*

recordIndex 134

recordIndex method 30

recordLength 134

recordLength method 30

registerRecordIndex 30, 134

registerRecordIndex method 30

rewriteRecord 135

rewriteRecord method 31

SearchCriterion 138

setAccess 135

setEmptyOnOpen 135

setStatus 135

Status 138

type 136

unlockRecord 136

writeRecord 136

writeRecord method 30

lccFile constructors

Constructor 129

in lccFile class 129

lccFile::readRecord

in Scope of data in lccBuf reference returned from 'read' methods 63

lccFileId

in Base classes 17

in File control 29

in lccFileId class 139

in Resource identification classes 18

lccFileId class

Constructor 139

operator= 139

overview 17, 29

reading records 29

lccFileId constructors

Constructor 139

in lccFileId class 139

lccFileIterator

in Browsing records 32

in Buffer objects 25

in Example of file control 32, 33

in File control 29

in lccFileIterator class 141

lccFileIterator class

Constructor 141

overview 29

readNextRecord 141

readNextRecord method 32

readPreviousRecord 32, 142

reset 142

lccFileIterator constructor

Constructor 141

in lccFileIterator class 141

lccGroupId

in lccGroupId class 145

lccGroupId class

Constructor 145

operator= 145

lccGroupId constructors

Constructor 145

in lccGroupId class 145

- lccJournal
 - in Buffer objects 25
 - in lccJournal class 147
 - in Object level 57
- lccJournal class
 - clearPrefix 147
 - Constructor 147
 - journalTypeeld 148
 - Options 150
 - put 148
 - registerPrefix 148
 - setJournalTypeeld 148
 - setPrefix 148
 - wait 148
 - writeRecord 149
- lccJournal constructors
 - Constructor 147
 - in lccJournal class 147
- lccJournalld
 - in lccJournalld class 151
- lccJournalld class
 - Constructor 151
 - number 151
 - operator= 151
- lccJournalld constructors
 - Constructor 151
 - in lccJournalld class 151
- lccJournalTypeeld
 - in Foundation Classes—reference 66
 - in lccJournalTypeeld class 153
- lccJournalTypeeld class
 - Constructor 153
 - operator= 153
- lccJournalTypeeld constructors
 - Constructor 153
 - in lccJournalTypeeld class 153
- lccKey
 - in Browsing records 32
 - in Deleting normal records 31
 - in File control 29
 - in lccKey class 155
 - in lccRecordIndex class 173
 - in Reading KSDS records 30
 - in Reading records 29
 - in Writing KSDS records 31
 - in Writing records 30
- lccKey class 30
 - assign 155
 - completeLength 155
 - Constructor 155
 - kind 155
 - Kind 157
 - operator!= 156
 - operator= 156
 - operator== 156
 - reading records 29
 - setKind 156
 - value 157
- lccKey constructors
 - Constructor 155
 - in lccKey class 155
- lccLockld
 - in lccLockld class 159
- lccLockld class
 - Constructor 159
 - operator= 159
- lccLockld constructors
 - Constructor 159
 - in lccLockld class 159
- lccMessage
 - in lccMessage class 161
 - in Support Classes 21
- lccMessage class
 - className 161
 - Constructor 161
 - methodName 161
 - number 161
 - summary 162
 - text 162
- lccMessage constructor
 - Constructor 161
 - in lccMessage class 161
- lccPartnerld
 - in lccPartnerld class 163
- lccPartnerld class
 - Constructor 163
 - operator= 163
- lccPartnerld constructors
 - Constructor 163
 - in lccPartnerld class 163
- lccProgram
 - in Buffer objects 25
 - in lccProgram class 165
 - in Program control 34, 35
 - in Resource classes 19
- lccProgram class
 - address 165
 - clearInputMessage 165
 - CommitOpt 168
 - Constructor 165
 - entryPoint 166
 - length 166
 - link 166
 - load 166
 - LoadOpt 168
 - program control 34
 - setInputMessage 167
 - unload 167
- lccProgram constructors
 - Constructor 165
 - in lccProgram class 165
- lccProgramld
 - in lccProgramld class 169
 - in Resource identification classes 18
- lccProgramld class
 - Constructor 169
 - operator= 169
- lccProgramld constructors
 - Constructor 169
 - in lccProgramld class 169
- lccRBA
 - in Browsing records 32

- lccRBA (*continued*)
 - in File control 29
 - in lccRBA class 171
 - in lccRecordIndex class 173
 - in Reading ESDS records 30
 - in Reading records 29
 - in Writing ESDS records 31
 - in Writing records 30
 - in Writing RRDS records 31
- lccRBA class
 - Constructor 171
 - number 172
 - operator!= 171, 172
 - operator= 171
 - operator== 171
 - reading records 29
- lccRBA constructor
 - Constructor 171
 - in lccRBA class 171
- lccRecordIndex
 - in C++ Exceptions and the Foundation Classes 53
 - in lccRecordIndex class 173
- lccRecordIndex class
 - Constructor 173
 - length 173
 - type 173
 - Type 174
- lccRecordIndex constructor (protected)
 - Constructor 173
 - in lccRecordIndex class 173
- lccRequestId
 - in Example of starting transactions 37, 38
 - in lccRequestId class 175
 - in Parameter passing conventions 63
- lccRequestId class
 - Constructor 175
 - operator= 175
- lccRequestId constructors
 - Constructor 175
 - in lccRequestId class 175
- lccResource
 - in Base classes 17
 - in Example of polymorphic behavior 61
 - in lccResource class 177
 - in Polymorphic Behavior 60
 - in Resource classes 19, 20
 - in Scope of data in lccBuf reference returned from 'read' methods 63
- lccResource class
 - actionOnCondition 177
 - ActionOnCondition 182
 - actionOnConditionAsChar 177
 - actionsOnConditionsText 178
 - clear 178
 - condition 178
 - conditionText 178
 - ConditionType 182
 - Constructor 177
 - get 178
 - handleEvent 179
 - HandleEventReturnOpt 182
- lccResource class (*continued*)
 - id 179
 - isEDFOn 179
 - isRouteOptionOn 179
 - name 179
 - overview 17
 - put 179
 - routeOption 180
 - setActionOnAnyCondition 180
 - setActionOnCondition 180
 - setActionsOnConditions 180
 - setEDF 180
 - setRouteOption 181
 - working with subclasses 26
- lccResource constructor (protected)
 - Constructor 177
 - in lccResource class 177
- lccResourceId
 - in Base classes 17
 - in C++ Exceptions and the Foundation Classes 53
 - in Resource identification classes 18
- lccResourceId class
 - Constructor 183
 - name 183
 - nameLength 183
 - operator= 184
 - overview 17, 18
- lccResourceId constructors (protected)
 - Constructor 183
 - in lccResourceId class 183
- lccRRN
 - in Browsing records 32
 - in Deleting normal records 31
 - in File control 29
 - in lccRecordIndex class 173
 - in lccRRN class 185
 - in Reading records 29
 - in Reading RRDS records 30
 - in Writing records 30
- lccRRN class
 - Constructor 185
 - number 186
 - operator!= 185, 186
 - operator= 185
 - operator== 185
 - reading records 29
- lccRRN constructors
 - Constructor 185
 - in lccRRN class 185
- lccSemaphore class
 - Constructor 187
 - lifeTime 187
 - LifeTime 189
 - lock 187
 - LockType 189
 - tryLock 188
 - type 188
 - unlock 188
- lccSemaphore constructor
 - Constructor 187
 - in lccSemaphore class 187

- lccSession
 - in Buffer objects 25
- lccSession class
 - allocate 192
 - AllocateOpt 199
 - connectProcess 192
 - Constructor 191
 - converse 193
 - convId 193
 - errorCode 193
 - extractProcess 193
 - flush 194
 - free 194
 - get 194
 - isErrorSet 194
 - isNoDataSet 194
 - isSignalSet 194
 - issueAbend 194
 - issueConfirmation 195
 - issueError 195
 - issuePrepare 195
 - issueSignal 195
 - PIPList 195
 - process 195
 - put 196
 - receive 196
 - send 196
 - sendInvite 197
 - sendLast 197
 - SendOpt 199
 - state 198
 - StateOpt 199
 - stateText 198
 - syncLevel 198
 - SyncLevel 200
- lccSession constructor (protected)
 - Constructor 191
 - in lccSession class 191
- lccSession constructors (public)
 - Constructor 191
 - in lccSession class 191
- lccStartRequestQ
 - in Accessing start data 36
 - in Buffer objects 25
 - in Example of starting transactions 37, 38
 - in lccRequestId class 175
 - in lccStartRequestQ class 201
 - in Mapping EXEC CICS calls to Foundation Class methods 285
 - in Parameter passing conventions 63
 - in Singleton classes 22
 - in Starting transactions asynchronously 36
- lccStartRequestQ class
 - cancel 201
 - CheckOpt 206
 - clearData 201
 - Constructor 201
 - data 202
 - instance 202
 - overview 22
 - ProtectOpt 206
- lccStartRequestQ class (continued)
 - queueName 202
 - registerData 202
 - reset 202
 - retrieveData 202
 - RetrieveOpt 206
 - returnTermId 203
 - returnTransId 203
 - setData 203
 - setQueueName 203
 - setReturnTermId 203, 204
 - setReturnTransId 204
 - setStartOpts 204
 - start 204
- lccStartRequestQ constructor (protected)
 - Constructor 201
 - in lccStartRequestQ class 201
- lccSysId
 - in lccSysId class 207
 - in Program control 35
- lccSysId class
 - Constructor 207
 - operator= 207
- lccSysId constructors
 - Constructor 207
 - in lccSysId class 207
- lccSystem
 - in Singleton classes 22
- lccSystem class
 - appName 209
 - beginBrowse 209
 - Constructor 209
 - dateFormat 210
 - endBrowse 210
 - freeStorage 210
 - getFile 210
 - getNextFile 211
 - getStorage 211
 - instance 211
 - operatingSystem 211
 - operatingSystemLevel 212
 - overview 22
 - release 212
 - releaseText 212
 - ResourceType 213
 - sysId 212
 - workArea 212
- lccSystem constructor (protected)
 - Constructor 209
 - in lccSystem class 209
- lccTask
 - in C++ Exceptions and the Foundation Classes 52
 - in Example of starting transactions 38
 - in lccAlarmRequestId class 81
 - in lccTask class 215
 - in Parameter level 57
 - in Singleton classes 22
 - in Support Classes 21
- lccTask class
 - abend 215
 - abendData 215

- lccTask class *(continued)*
 - AbendDumpOpt 223
 - AbendHandlerOpt 222
 - commitUOW 215
 - Constructor 215
 - delay 216
 - dump 216
 - DumpOpts 223
 - enterTrace 217
 - facilityType 217
 - FacilityType 223
 - freeStorage 217
 - getStorage 217
 - instance 218
 - isCommandSecurityOn 218
 - isCommitSupported 218
 - isResourceSecurityOn 218
 - isRestarted 218
 - isStartDataAvailable 219
 - number 219
 - overview 22
 - principalSysId 219
 - priority 219
 - rollBackUOW 219
 - setDumpOpts 219
 - setPriority 220
 - setWaitText 220
 - startType 220
 - StartType 223
 - StorageOpts 224
 - suspend 220
 - TraceOpt 224
 - transId 220
 - triggerDataQueueId 220
 - userId 221
 - waitExternal 221
 - waitOnAlarm 221
 - WaitPostType 224
 - WaitPurgeability 224
 - workArea 222
- lccTask Constructor (protected)
 - Constructor 215
 - in lccTask class 215
- lccTask::commitUOW
 - in Scope of data in lccBuf reference returned from 'read' methods 63
- lccTempstore
 - in Working with lccResource subclasses 26
- lccTempStore
 - in Automatic condition handling (callHandleEvent) 55
 - in Buffer objects 25
 - in C++ Exceptions and the Foundation Classes 53
 - in Deleting items 42
 - in Example of polymorphic behavior 61
 - in Example of Temporary Storage 42
 - in lccTempStore class 225
 - in Reading items 41
 - in Resource classes 19
 - in Temporary storage 41
 - in Transient Data 39
- lccTempStore *(continued)*
 - in Updating items 42
 - in Working with lccResource subclasses 27
 - in Writing items 41
- lccTempStore class
 - clear 225
 - Constructor 225
 - empty 226
 - get 226
 - Location 228
 - NoSpaceOpt 228
 - numberOfItems 226
 - put 226
 - readItem 226
 - readNextItem 226
 - rewriteItem 227
 - writeItem 227
- lccTempStore constructors
 - Constructor 225
 - in lccTempStore class 225
- lccTempStore::readItem
 - in Scope of data in lccBuf reference returned from 'read' methods 63
- lccTempStore::readNextItem
 - in Scope of data in lccBuf reference returned from 'read' methods 63
- lccTempStoreId
 - in Base classes 17
 - in Example of Temporary Storage 42
 - in lccTempStoreId class 229
 - in Temporary storage 41
- lccTempStoreId class
 - Constructor 229
 - operator= 229
- lccTempStoreId constructors
 - Constructor 229
 - in lccTempStoreId class 229
- lccTermId
 - in Base classes 17
 - in C++ Exceptions and the Foundation Classes 53
 - in Example of starting transactions 37
 - in Example of terminal control 44
 - in lccTermId class 231
 - in Terminal control 43
- lccTermId class
 - Constructor 231
 - operator= 231
 - overview 17
- lccTermId constructors
 - Constructor 231
 - in lccTermId class 231
- lccTerminal
 - in Buffer objects 25
 - in Example of terminal control 44
 - in Finding out information about a terminal 44
 - in lccTerminalData class 247
 - in Receiving data from a terminal 44
 - in Resource classes 19, 20
 - in Sending data to a terminal 43
 - in Singleton classes 22
 - in Terminal control 43

- lccTerminal class
 - AID 233
 - AIDVal 244
 - Case 244
 - clear 233
 - Color 244
 - Constructor 233
 - cursor 233
 - data 233
 - erase 234
 - freeKeyboard 234
 - get 234
 - height 234
 - Highlight 245
 - inputCursor 234
 - instance 234
 - line 235
 - netName 235
 - NextTransIdOpt 245
 - operator<< 235, 236, 237
 - put 237
 - receive 237
 - receive3270Data 237
 - registerInputMessage 167
 - send 238
 - send3270Data 239
 - sendLine 239, 240
 - setColor 240
 - setCursor 240, 241
 - setHighlight 241
 - setLine 241
 - setNewLine 241
 - setNextCommArea 241
 - setNextInputMessage 242
 - setNextTransId 242
 - signoff 242
 - signon 242
 - waitForAID 243
 - width 243
 - workArea 243
- lccTerminal constructor (protected)
 - Constructor 233
 - in lccTerminal class 233
- lccTerminal::receive
 - in Scope of data in lccBuf reference returned from 'read' methods 63
- lccTerminalData
 - in Example of terminal control 44
 - in Finding out information about a terminal 44
 - in lccTerminalData class 247
 - in Terminal control 43
- lccTerminalData class
 - alternateHeight 247
 - alternateWidth 247
 - Constructor 247
 - defaultHeight 248
 - defaultWidth 248
 - graphicCharCodeSet 248
 - graphicCharSetId 248
 - isAPLKeyboard 248
 - isAPLText 248
- lccTerminalData class (continued)
 - isBTrans 249
 - isColor 249
 - isEWA 249
 - isExtended3270 249
 - isFieldOutline 249
 - isGoodMorning 250
 - isHighlight 250
 - isKatakana 250
 - isMSRControl 250
 - isPS 250
 - isSOSI 250
 - isTextKeyboard 251
 - isTextPrint 251
 - isValidation 251
- lccTerminalData constructor (protected)
 - Constructor 247
 - in lccTerminalData class 247
- lccTime
 - in Base classes 18
 - in lccTime class 253
 - in Parameter passing conventions 63
 - in Support Classes 21
- lccTime class
 - Constructor 253
 - hours 253
 - minutes 253
 - overview 18
 - seconds 253
 - timeInHours 253
 - timeInMinutes 254
 - timeInSeconds 254
 - type 254
 - Type 255
- lccTime constructor (protected)
 - Constructor 253
 - in lccTime class 253
- lccTimeInterval
 - in Base classes 18
 - in delay 216
 - in Example of starting transactions 37, 38
 - in lccTime class 253
 - in Support Classes 21
- lccTimeInterval class
 - Constructor 257
 - operator= 257
 - set 257
- lccTimeInterval constructors
 - Constructor 257
 - in lccTimeInterval class 257
- lccTimeOfDay
 - in Base classes 18
 - in delay 216
 - in lccTime class 253
 - in Support Classes 21
- lccTimeOfDay class
 - Constructor 259
 - operator= 259
 - set 259
- lccTimeOfDay constructors
 - Constructor 259

- lccTimeOfDay constructors *(continued)*
 - in lccTimeOfDay class 259
- lccTPNameId
 - in lccTPNameId class 261
- lccTPNameId class
 - Constructor 261
 - operator= 261
- lccTPNameId constructors
 - Constructor 261
 - in lccTPNameId class 261
- lccTransId
 - in Base classes 17
 - in Example of starting transactions 37
 - in lccResourceId class 183
 - in lccTransId class 263
 - in Parameter passing conventions 63
- lccTransId class
 - Constructor 263
 - operator= 263
 - overview 17
- lccTransId constructors
 - Constructor 263
 - in lccTransId class 263
- lccUser class
 - changePassword 265
 - Constructor 265
 - daysUntilPasswordExpires 266
 - ESMReason 266
 - ESMResponse 266
 - groupId 266
 - invalidPasswordAttempts 266
 - language 266
 - lastPasswordChange 266
 - lastUseTime 266
 - passwordExpiration 267
 - setLanguage 267
 - verifyPassword 267
- lccUser constructors
 - Constructor 265
 - in lccUser class 265
- lccUserControl
 - in C++ Exceptions and the Foundation Classes 52
 - in Example of file control 32
 - in Example of managing transient data 40
 - in Example of polymorphic behavior 60
 - in Example of starting transactions 37
 - in Example of Temporary Storage 42
 - in Example of terminal control 44
 - in Example of time and date services 46
 - in Hello World 9
 - in main function 275
 - in Program control 34
 - in Singleton classes 22
- lccUserControl class 9
- lccUserId
 - in lccUserId class 269
- lccUserId class
 - Constructor 269
 - operator= 269
- lccUserId constructors
 - Constructor 269
- lccUserId constructors *(continued)*
 - in lccUserId class 269
- lccValue
 - in Foundation Classes—reference 66
- lccValue structure
 - CVDA 271
- id
 - in lccResource class 179
- Id
 - in Resource identification classes 18
- id (parameter)
 - in Constructor 81, 117, 121, 129, 139, 145, 147, 151, 153, 159, 163, 165, 169, 175, 183, 187, 191, 207, 225, 229, 231, 261, 263, 265, 269
 - in getFile 210
 - in operator= 82, 115, 121, 139, 145, 151, 153, 159, 163, 169, 175, 184, 207, 229, 231, 261, 263, 269
 - in setJournalTypeId 148
 - in signon 242
 - in waitOnAlarm 222
- ifSOSReturnCondition
 - in StorageOpts 224
- ignoreAbendHandler
 - in AbendHandlerOpt 223
- immediate
 - in NextTransIdOpt 245
- index (parameter)
 - in Constructor 129, 141
 - in registerRecordIndex 134
 - in reset 142
- Inherited protected methods
 - in lccAbendData class 75
 - in lccAbsTime class 80
 - in lccAlarmRequestId class 82
 - in lccBuf class 96
 - in lccClock class 102
 - in lccConsole class 109
 - in lccControl class 114
 - in lccConvId class 116
 - in lccDataQueue class 119
 - in lccDataQueueId class 122
 - in lccEvent class 124
 - in lccException class 127
 - in lccFile class 137
 - in lccFileId class 140
 - in lccFileIterator class 143
 - in lccGroupId class 146
 - in lccJournal class 150
 - in lccJournalId class 152
 - in lccJournalTypeId class 154
 - in lccKey class 157
 - in lccLockId class 160
 - in lccMessage class 162
 - in lccPartnerId class 164
 - in lccProgram class 168
 - in lccProgramId class 170
 - in lccRBA class 172
 - in lccRecordIndex class 174
 - in lccRequestId class 176
 - in lccResource class 181
 - in lccResourceId class 184

Inherited protected methods *(continued)*

- in lccRRN class 186
- in lccSemaphore class 189
- in lccSession class 199
- in lccStartRequestQ class 206
- in lccSysId class 208
- in lccSystem class 213
- in lccTask class 222
- in lccTempStore class 228
- in lccTempStoreId class 230
- in lccTermId class 232
- in lccTerminal class 244
- in lccTerminalData class 252
- in lccTime class 254
- in lccTimeInterval class 258
- in lccTimeOfDay class 260
- in lccTPNameId class 262
- in lccTransId class 264
- in lccUser class 268
- in lccUserId class 270

Inherited public methods

- in lccAbendData class 74
- in lccAbsTime class 80
- in lccAlarmRequestId class 82
- in lccBuf class 96
- in lccClock class 101
- in lccConsole class 109
- in lccControl class 114
- in lccConvId class 115
- in lccDataQueue class 118
- in lccDataQueueId class 121
- in lccEvent class 124
- in lccException class 127
- in lccFile class 136
- in lccFileId class 139
- in lccFileIterator class 142
- in lccGroupId class 145
- in lccJournal class 149
- in lccJournalId class 152
- in lccJournalTypeId class 153
- in lccKey class 157
- in lccLockId class 159
- in lccMessage class 162
- in lccPartnerId class 163
- in lccProgram class 167
- in lccProgramId class 169
- in lccRBA class 172
- in lccRecordIndex class 173
- in lccRequestId class 176
- in lccResource class 181
- in lccResourceId class 184
- in lccRRN class 186
- in lccSemaphore class 188
- in lccSession class 198
- in lccStartRequestQ class 205
- in lccSysId class 207
- in lccSystem class 213
- in lccTask class 222
- in lccTempStore class 228
- in lccTempStoreId class 229
- in lccTermId class 231

Inherited public methods *(continued)*

- in lccTerminal class 244
- in lccTerminalData class 251
- in lccTime class 254
- in lccTimeInterval class 258
- in lccTimeOfDay class 260
- in lccTPNameId class 261
- in lccTransId class 263
- in lccUser class 267
- in lccUserId class 269
- initByte (parameter)
 - in getStorage 211, 217
- initData
 - in lccControl class 112
 - in Public methods 112
- initializeEnvironment
 - in Functions 67
 - in lcc structure 67
 - in Method level 57
 - in Storage management 62
- initRBA (parameter)
 - in Constructor 171
- initRRN (parameter)
 - in Constructor 185
- initValue (parameter)
 - in Constructor 155
- inputCursor
 - in lccTerminal class 234
- insert
 - in Example of Temporary Storage 43
 - in lccBuf class 92
 - in lccBuf constructors 26
- Installed contents
 - Location 6
- instance
 - in lccAbendData class 74
 - in lccConsole class 107
 - in lccControl class 112
 - in lccStartRequestQ class 202
 - in lccSystem class 211
 - in lccTask class 218
 - in lccTerminal class 234
 - in Singleton classes 22
- internal
 - in DataAreaOwner 97
- internalError
 - in C++ Exceptions and the Foundation Classes 53
 - in Type 128
- interval (parameter)
 - in setReplyTimeout 108
- invalidArgument
 - in C++ Exceptions and the Foundation Classes 53
 - in Type 127
- invalidMethodCall
 - in C++ Exceptions and the Foundation Classes 53
 - in Type 127
- invalidPasswordAttempts
 - in lccUser class 266
- IPMD 50
- isAddable
 - in lccFile class 131

- isAddable (*continued*)
 - in Writing ESDS records 31
 - in Writing KSDS records 31
 - in Writing RRDS records 31
- isAPLKeyboard
 - in lccTerminalData class 248
 - in Public methods 248
- isAPLText
 - in lccTerminalData class 248
 - in Public methods 248
- isBrowsable
 - in lccFile class 131
- isBTrans
 - in lccTerminalData class 249
- isClassMemoryMgmtOn
 - in Functions 68
 - in lcc structure 68
- isColor
 - in lccTerminalData class 249
- isCommandSecurityOn
 - in lccTask class 218
- isCommitSupported
 - in lccTask class 218
- isCreated
 - in lccControl class 112
- isDeletable
 - in lccFile class 131
- isDumpAvailable
 - in lccAbendData class 74
- isEDFOn
 - in Functions 68
 - in lcc structure 68
 - in lccResource class 179
- isEmptyOnOpen
 - in lccFile class 132
- isErrorSet
 - in lccSession class 194
- isEWA
 - in lccTerminalData class 249
- isExpired
 - in lccAlarmRequestId class 81
- isExtended3270
 - in lccTerminalData class 249
 - in Public methods 249
- isFamilySubsetEnforcementOn
 - in Functions 68
 - in lcc structure 68
- isFieldOutline
 - in lccTerminalData class 249
 - in Public methods 249
- isFMHContained
 - in lccBuf class 92
 - in Public methods 92
- isGoodMorning
 - in lccTerminalData class 250
 - in Public methods 250
- isHighlight
 - in lccTerminalData class 250
- isKatakana
 - in lccTerminalData class 250
- isMSRControl
 - in lccTerminalData class 250
- isNoDataSet
 - in lccSession class 194
- isPS
 - in lccTerminalData class 250
- ISR2
 - in Example of starting transactions 37
- isReadable
 - in lccFile class 132
 - in Reading ESDS records 30
 - in Reading KSDS records 30
 - in Reading RRDS records 30
- isReadable method 30
- isRecoverable
 - in lccFile class 132
- isResourceSecurityOn
 - in lccTask class 218
- isRestarted
 - in lccTask class 218
- isRouteOptionOn
 - in lccResource class 179
 - in Public methods 179
- isSignalSet
 - in lccSession class 194
- isSOSI
 - in lccTerminalData class 250
- isStartDataAvailable
 - in lccTask class 219
- issueAbend
 - in lccSession class 194
- issueConfirmation
 - in lccSession class 195
- issueError
 - in lccSession class 195
- issuePrepare
 - in lccSession class 195
- issueSignal
 - in lccSession class 195
- isTextKeyboard
 - in lccTerminalData class 251
 - in Public methods 251
- isTextPrint
 - in lccTerminalData class 251
 - in Public methods 251
- isUpdatable
 - in lccFile class 132
- isValidation
 - in lccTerminalData class 251
- item (parameter)
 - in rewriteItem 227
 - in writeItem 118, 227
- itemNum (parameter)
 - in readItem 226
 - in rewriteItem 227
- ITMP
 - in Example of starting transactions 37

J

journalNum (parameter)
 in Constructor 147, 151
 in operator= 151
journalTypeld
 in lccJournal class 148
journalTypeName (parameter)
 in Constructor 153
 in operator= 153
jtypeid (parameter)
 in setJournalTypeld 148

K

key
 complete 30
 generic 30
key (parameter)
 in Constructor 155
 in Example of file control 33
 in operator!= 156
 in operator= 156
 in operator== 156
keyLength
 in lccFile class 132
 in Reading KSDS records 30
 in Writing KSDS records 31
keyLength method 30
keyPosition
 in lccFile class 133
 in Reading KSDS records 30
 in writing KSDS records 31
keyPosition method 30
kind
 in lccKey class 155
Kind
 in Enumerations 157
 in lccKey class 157
kind (parameter)
 in Constructor 155
 in setKind 157
KSDS
 in File control 29
KSDS file 29

L

language
 in lccUser class 266
language (parameter)
 in setLanguage 267
lastCommand
 in StateOpt 199
lastPasswordChange
 in lccUser class 266
lastUseTime
 in lccUser class 266
length
 in lccProgram class 166
 in lccRecordIndex class 173

length (parameter)
 in append 90
 in assign 90, 155
 in Constructor 89
 in cut 91
 in insert 92
 in overlay 95
 in replace 96
 in setDataLength 96
level (parameter)
 in connectProcess 192
level0
 in SyncLevel 200
level1
 in SyncLevel 200
level2
 in SyncLevel 200
life (parameter)
 in Constructor 187
lifeTime
 in lccSemaphore class 187
LifeTime
 in Enumerations 189
 in lccSemaphore class 189
line
 in Finding out information about a terminal 44
 in lccTerminal class 235
lineNum (parameter)
 in setLine 241
link
 in lccProgram class 166
load
 in lccProgram class 166
LoadOpt
 in Enumerations 168
 in lccProgram class 168
loc (parameter)
 in Constructor 225
Location
 in Dynamic link library 6
 in Enumerations 228
 in Header files 6
 in lccTempStore class 228
 in Installed contents 6
 in Sample source code 6
lock
 in lccSemaphore class 187
LockType
 in Enumerations 189
 in lccSemaphore class 189

M

main
 in C++ Exceptions and the Foundation Classes 51
 in Example of file control 32
 in Example of managing transient data 40
 in Example of polymorphic behavior 60
 in Example of starting transactions 37
 in Example of Temporary Storage 42
 in Example of terminal control 44

- main (*continued*)
 - in Example of time and date services 45
 - in Header files 6
 - in main function 275
 - in Program control 34
 - in Storage management 62
- main function
 - in Hello World 9
- majorCode
 - in ConditionType 182
- manual
 - in UpdateMode 103
- Manual condition handling (noAction)
 - in CICS conditions 54
 - in Conditions, errors, and exceptions 54
- maxValue
 - in Range 105
- mem (parameter)
 - in initializeEnvironment 68
- memory
 - in Location 228
- message
 - in lccException class 126
- message (parameter)
 - in Constructor 125
 - in setNextInputMessage 242
- method
 - in Foundation Classes—reference 66
- Method level
 - in Conditions, errors, and exceptions 57
 - in Platform differences 57
- methodName
 - in lccEvent class 124
 - in lccException class 126
 - in lccMessage class 161
- methodName (parameter)
 - in Constructor 123, 125, 161
- milliseconds
 - in lccAbsTime class 78
 - in lccClock class 100
- minorCode
 - in ConditionType 182
- minutes
 - in lccAbsTime class 78
 - in lccTime class 253
- minutes (parameter)
 - in Constructor 253, 257, 259
 - in set 257, 259
- Miscellaneous
 - Example of polymorphic behavior 60
- mixed
 - in Case 244
- mode (parameter)
 - in readNextRecord 141
 - in readPreviousRecord 142
 - in readRecord 133
- monthOfYear
 - in Example of time and date services 46
 - in lccAbsTime class 78
 - in lccClock class 100

- MonthOfYear
 - in Enumerations 102
 - in lccClock class 102
- msg (parameter)
 - in clearInputMessage 165
 - in registerInputMessage 167
 - in setInputMessage 167
- MVS/ESA
 - in ClassMemoryMgmt 69
 - in Storage management 62
- MVSPost
 - in WaitPostType 224
- MyTempStore
 - in Automatic condition handling (callHandleEvent) 55

N

- N
 - in operatingSystem 211
- name
 - in lccResource class 179
 - in lccResourceId class 183
- name (parameter)
 - in Constructor 81, 159, 207, 229, 231, 261, 263, 269
 - in operator= 159, 207, 229, 231, 261, 263, 269
 - in setWaitText 220
- nameLength
 - in lccResourceId class 183
- NameOpt
 - in Enumerations 87
 - in lccBase class 87
- netName
 - in lccTerminal class 235
- neutral
 - in Color 244
- new
 - in Storage management 62
- new operator 15
- newPassword (parameter)
 - in changePassword 265
 - in signon 242, 243
- NextTransIdOpt
 - in Enumerations 245
 - in lccTerminal class 245
- noAccess
 - in Access 137
- noAction
 - in ActionOnCondition 182
 - in CICS conditions 54
- noCommitOnReturn
 - in CommitOpt 168
- NONCICS
 - in ASRAKeyType 72
- none
 - in FacilityType 223
- noQueue
 - in AllocateOpt 199
- normal
 - in ReadMode 137

- normal (*continued*)
 - in SendOpt 199
 - in TraceOpt 224
- NoSpaceOpt
 - in Enumerations 228
 - in lccTempStore class 228
- noSuspend
 - in Options 150
- notAddable
 - in Access 137
- NOTAPPLIC
 - in ASRAKeyType 72
 - in ASRASpaceType 73
 - in ASRAStorageType 73
- notBrowsable
 - in Access 137
- notDeletable
 - in Access 137
- notPurgeable
 - in WaitPurgeability 224
- notReadable
 - in Access 137
- notUpdatable
 - in Access 137
- num (parameter)
 - in operator!= 172
 - in operator<< 94, 95, 236, 237
 - in operator= 171, 185
 - in operator== 171
- number
 - in lccException class 126
 - in lccJournalId class 151
 - in lccMessage class 161
 - in lccRBA class 172
 - in lccRRN class 186
 - in lccTask class 219
 - in Writing RRDS records 31
- number (parameter)
 - in Constructor 161
 - in setCustomClassNum 86
- numberOfItems
 - in lccTempStore class 226
- numEvents (parameter)
 - in waitExternal 221
- numLines (parameter)
 - in setNewLine 241
- numRoutes (parameter)
 - in setRouteCodes 108

O

- obj (parameter)
 - in Using an object 16
- object
 - creating 15
 - deleting 16
 - in GetOpt 70
 - using 16
- object (parameter)
 - in Constructor 123, 125
 - in operator delete 86

- Object level
 - in Conditions, errors, and exceptions 56
 - in Platform differences 56
- objectCreationError
 - in C++ Exceptions and the Foundation Classes 52
 - in Type 127
- offset (parameter)
 - in cut 91
 - in dataArea 91
 - in insert 92
 - in replace 96
 - in setCursor 240
- onOff (parameter)
 - in setEDF 69, 180
- open
 - in Status 138
- openStatus
 - in lccFile class 133
- operatingSystem
 - in lccSystem class 211
 - in Public methods 211
- operatingSystemLevel
 - in lccSystem class 212
- operator const char*
 - in lccBuf class 92
- operator delete
 - in lccBase class 86
 - in Public methods 86
- operator new
 - in lccBase class 86
- operator!=
 - in lccBuf class 93
 - in lccKey class 156
 - in lccRBA class 171, 172
 - in lccRRN class 185, 186
 - in Public methods 93
- operator<<
 - in lccBuf class 93, 95
 - in lccTerminal class 235, 236, 237
 - in Working with lccResource subclasses 27
- operator+=
 - in lccBuf class 93
- operator=
 - in Example of file control 33
 - in lccAbsTime class 79
 - in lccAlarmRequestId class 82
 - in lccBuf class 92
 - in lccConvId class 115
 - in lccDataQueueId class 121
 - in lccFileId class 139
 - in lccGroupId class 145
 - in lccJournalId class 151
 - in lccJournalTypeId class 153
 - in lccKey class 156
 - in lccLockId class 159
 - in lccPartnerId class 163
 - in lccProgramId class 169
 - in lccRBA class 171
 - in lccRequestId class 175
 - in lccResourceId class 184
 - in lccRRN class 185

- operator= (continued)
 - in lccSysId class 207
 - in lccTempStoreId class 229
 - in lccTermId class 231
 - in lccTimeInterval class 257
 - in lccTimeOfDay class 259
 - in lccTPNameId class 261
 - in lccTransId class 263
 - in lccUserId class 269
 - in Protected methods 184
 - in Public methods 79, 257
 - in Working with lccResource subclasses 27
- operator==
 - in lccBuf class 93
 - in lccKey class 156
 - in lccRBA class 171
 - in lccRRN class 185
- opt (parameter)
 - in abendCode 71
 - in access 130
 - in accessMethod 130
 - in alternateHeight 247
 - in alternateWidth 247
 - in ASRAInterrupt 71
 - in ASRAKeyType 72
 - in ASRAPSW 72
 - in ASRARegisters 72
 - in ASRASpaceType 73
 - in ASRAStorageType 73
 - in className 85
 - in defaultHeight 248
 - in defaultWidth 248
 - in enableStatus 131
 - in enterTrace 217
 - in graphicCharCodeSet 248
 - in graphicCharSetId 248
 - in height 234
 - in isAddable 131
 - in isAPLKeyboard 248
 - in isAPLText 248
 - in isBrowsable 131
 - in isBTrans 249
 - in isColor 249
 - in isDeletable 131
 - in isDumpAvailable 74
 - in isEmptyOnOpen 132
 - in isEWA 249
 - in isExtended3270 249
 - in isFieldOutline 249
 - in isGoodMorning 250
 - in isHighlight 250
 - in isKatakana 250
 - in isMSRControl 250
 - in isPS 250
 - in isReadable 132
 - in isRecoverable 132
 - in isSOSI 251
 - in isTextKeyboard 251
 - in isTextPrint 251
 - in isUpdatable 132
 - in isValidation 251
- opt (parameter) (continued)
 - in keyLength 132
 - in keyPosition 133
 - in link 166
 - in load 167
 - in openStatus 133
 - in originalAbendCode 74
 - in principalSysId 219
 - in priority 219
 - in programName 74
 - in recordFormat 134
 - in recordLength 134
 - in rewriteItem 227
 - in setNextTransId 242
 - in type 136
 - in userId 221
 - in waitExternal 221
 - in width 243
 - in write 108, 109
 - in writeAndGetReply 109
 - in writeItem 227
- opt1 (parameter)
 - in abend 215
- opt2 (parameter)
 - in abend 215
- option (parameter)
 - in allocate 192
 - in retrieveData 203
 - in send 196
 - in sendInvite 197
 - in sendLast 197
 - in state 198
 - in stateText 198
 - in wait 149
 - in writeRecord 149
- Options
 - in Enumerations 150
 - in lccJournal class 150
- options (parameter)
 - in Constructor 147
- opts (parameter)
 - in setDumpOpts 220
- originalAbendCode
 - in lccAbendData class 74
- OS/2
 - in ClassMemoryMgmt 69
 - in Storage management 62
- OS/2, CICS
 - in Platform differences 56
- Other datasets for CICS/ESA
 - in Installed contents 6
- Output from sample programs
 - First Screen 294
 - Second Screen 294
- overlay
 - in lccBuf class 95
- overview of Foundation Classes 17
- Overview of the foundation classes
 - Calling methods on a resource object 22
 - Creating a resource object 21

P

P

- in operatingSystem 211
- PA1 to PA3
 - in AIDVal 244
- packedDecimal
 - in lccAbsTime class 79
- Parameter level
 - in Conditions, errors, and exceptions 57
 - in Platform differences 57
- parameter passing 62
- Parameter passing conventions
 - in Miscellaneous 62
- partnerName (parameter)
 - in Constructor 163
 - in operator= 163
- password (parameter)
 - in changePassword 265
 - in signon 242, 243
 - in verifyPassword 267
- passwordExpiration
 - in lccUser class 267
- PF1 to PF24
 - in AIDVal 244
- pink
 - in Color 244
- PIP (parameter)
 - in connectProcess 192, 193
- PIPList
 - in lccSession class 195
- platform differences
 - method level 57
 - object level 56
 - parameter level 57
- Platform differences
 - in Conditions, errors, and exceptions 56
 - Method level 57
 - Object level 56
 - Parameter level 57
- platformError
 - in Type 128
- Platforms
 - in Enumerations 70
 - in lcc structure 70
- polymorphic behavior 59
- Polymorphic Behavior
 - Example of polymorphic behavior 60
 - in Miscellaneous 59
- popt (parameter)
 - in setStartOpts 204
- prefix (parameter)
 - in registerPrefix 148
 - in setPrefix 148
- pri (parameter)
 - in setPriority 220
- principalSysId
 - in lccTask class 219
 - in Public methods 219
- print
 - in Polymorphic Behavior 60
- priority
 - in lccTask class 219
 - in Public methods 219
- process
 - in lccSession class 195
- profile (parameter)
 - in Constructor 191
- progName (parameter)
 - in Constructor 165, 169
 - in operator= 169
- program control
 - example 34
 - introduction 34
- Program control
 - in Using CICS Services 34
- programId
 - in lccControl class 112
 - in Method level 57
 - in Public methods 112
- programId (parameter)
 - in setAbendHandler 113
- programName
 - in lccAbendData class 74
 - in Public methods 74
- programName (parameter)
 - in setAbendHandler 113
- Protected methods
 - in lccBase class 86
 - in lccResourceId class 184
 - operator= 184
 - setClassName 86
 - setCustomClassNum 86
- ProtectOpt
 - in Enumerations 206
 - in lccStartRequestQ class 206
- pStorage (parameter)
 - in freeStorage 210
- Public methods
 - abend 215
 - abendCode 71
 - abendData 215
 - absTime 99
 - access 129
 - accessMethod 130
 - actionOnCondition 177
 - actionOnConditionAsChar 177
 - actionsOnConditionsText 178
 - address 165
 - AID 233
 - allocate 192
 - alternateHeight 247
 - alternateWidth 247
 - append 90
 - applName 209
 - ASRAInterrupt 71
 - ASRAKeyType 72
 - ASRAPSW 72
 - ASRARegisters 72
 - ASRASpaceType 73
 - ASRAStorageType 73
 - assign 90, 91, 155

Public methods *(continued)*

beginBrowse 209
beginInsert(VSAM only) 130
callingProgramId 111
cancel 201
cancelAbendHandler 111
cancelAlarm 99
changePassword 265
className 85, 123, 126, 161
classType 85, 123, 126
clear 117, 178, 225, 233
clearData 201
clearInputMessage 165
clearPrefix 147
commArea 111
commitUOW 215
completeLength 155
condition 123, 178
conditionText 124, 178
connectProcess 192
console 112
converse 193
convId 193
cursor 233
customClassNum 86
cut 91
data 202, 233
dataArea 91
dataAreaLength 91
dataAreaOwner 91
dataAreaType 91
dataLength 92
date 77, 99
dateFormat 210
dayOfMonth 77, 100
dayOfWeek 78, 100
daysSince1900 78, 100
daysUntilPasswordExpires 266
defaultHeight 248
defaultWidth 248
delay 216
deleteLockedRecord 130
deleteRecord 130
dump 216
empty 117, 226
enableStatus 131
endBrowse 210
endInsert(VSAM only) 131
enterTrace 217
entryPoint 166
erase 234
errorCode 193
ESMReason 266
ESMResponse 266
extractProcess 193
facilityType 217
flush 194
free 194
freeKeyboard 234
freeStorage 210, 217
get 117, 178, 194, 226, 234

Public methods *(continued)*

getFile 210
getNextFile 211
getStorage 211, 217
graphicCharCodeSet 248
graphicCharSetId 248
groupId 266
handleEvent 179
height 234
hours 78, 253
id 179
in lccAbendData class 71
in lccAbsTime class 77
in lccAlarmRequestId class 81
in lccBase class 85
in lccBuf class 90
in lccClock class 99
in lccConsole class 107
in lccControl class 111
in lccConvId class 115
in lccDataQueue class 117
in lccDataQueueId class 121
in lccEvent class 123
in lccException class 126
in lccFile class 129
in lccFileId class 139
in lccFileIterator class 141
in lccGroupId class 145
in lccJournal class 147
in lccJournalId class 151
in lccJournalTypeId class 153
in lccKey class 155
in lccLockId class 159
in lccMessage class 161
in lccPartnerId class 163
in lccProgram class 165
in lccProgramId class 169
in lccRBA class 171
in lccRecordIndex class 173
in lccRequestId class 175
in lccResource class 177
in lccResourceId class 183
in lccRRN class 185
in lccSemaphore class 187
in lccSession class 192
in lccStartRequestQ class 201
in lccSysId class 207
in lccSystem class 209
in lccTask class 215
in lccTempStore class 225
in lccTempStoreId class 229
in lccTermId class 231
in lccTerminal class 233
in lccTerminalData class 247
in lccTime class 253
in lccTimeInterval class 257
in lccTimeOfDay class 259
in lccTPNameId class 261
in lccTransId class 263
in lccUser class 265
in lccUserId class 269

Public methods (*continued*)

initData 112
 inputCursor 234
 insert 92
 instance 74, 107, 112, 202, 211, 218, 234
 invalidPasswordAttempts 266
 isAddable 131
 isAPLKeyboard 248
 isAPLText 248
 isBrowsable 131
 isBTrans 249
 isColor 249
 isCommandSecurityOn 218
 isCommitSupported 218
 isCreated 112
 isDeletable 131
 isDumpAvailable 74
 isEDFOn 179
 isEmptyOnOpen 132
 isErrorSet 194
 isEWA 249
 isExpired 81
 isExtended3270 249
 isFieldOutline 249
 isFMHContained 92
 isGoodMorning 250
 isHighlight 250
 isKatakana 250
 isMSRControl 250
 isNoDataSet 194
 isPS 250
 isReadable 132
 isRecoverable 132
 isResourceSecurityOn 218
 isRestarted 218
 isRouteOptionOn 179
 isSignalSet 194
 isSOSI 250
 isStartDataAvailable 219
 issueAbend 194
 issueConfirmation 195
 issueError 195
 issuePrepare 195
 issueSignal 195
 isTextKeyboard 251
 isTextPrint 251
 isUpdatable 132
 isValidation 251
 journalTypeid 148
 keyLength 132
 keyPosition 133
 kind 155
 language 266
 lastPasswordChange 266
 lastUseTime 266
 length 166, 173
 lifeTime 187
 line 235
 link 166
 load 166
 lock 187

Public methods (*continued*)

message 126
 methodName 124, 126, 161
 milliSeconds 78, 100
 minutes 78, 253
 monthOfYear 78, 100
 name 179, 183
 nameLength 183
 netName 235
 number 126, 151, 161, 172, 186, 219
 numberOfItems 226
 openStatus 133
 operatingSystem 211
 operatingSystemLevel 212
 operator const char* 92
 operator delete 86
 operator new 86
 operator!= 93, 156, 171, 172, 185, 186
 operator<< 93, 95, 235, 236, 237
 operator+= 93
 operator= 79, 82, 92, 115, 121, 139, 145, 151, 153,
 156, 159, 163, 169, 171, 175, 185, 207, 229, 231,
 257, 259, 261, 263, 269
 operator== 93, 156, 171, 185
 originalAbendCode 74
 overlay 95
 packedDecimal 79
 passwordExpiration 267
 PIPList 195
 principalSysId 219
 priority 219
 process 195
 programId 112
 programName 74
 put 107, 118, 148, 179, 196, 226, 237
 queueName 202
 readItem 118, 226
 readNextItem 226
 readNextRecord 141
 readPreviousRecord 142
 readRecord 133
 receive 196, 237
 receive3270Data 237
 recordFormat 134
 recordIndex 134
 recordLength 134
 registerData 202
 registerInputMessage 167
 registerPrefix 148
 registerRecordIndex 134
 release 212
 releaseText 212
 replace 95
 replyTimeout 107
 reset 142, 202
 resetAbendHandler 112
 resetRouteCodes 108
 retrieveData 202
 returnProgramId 113
 returnTermId 203
 returnTransId 203

Public methods (*continued*)

- rewriteItem 227
- rewriteRecord 135
- rollBackUOW 219
- routeOption 180
- run 113
- seconds 79, 253
- send 196, 238
- send3270Data 239
- sendInvite 197
- sendLast 197
- sendLine 239, 240
- session 113
- set 257, 259
- setAbendHandler 113
- setAccess 135
- setActionOnAnyCondition 180
- setActionOnCondition 180
- setActionsOnConditions 180
- setAlarm 101
- setAllRouteCodes 108
- setColor 240
- setCursor 240, 241
- setData 203
- setDataLength 96
- setDumpOpts 219
- setEDF 180
- setEmptyOnOpen 135
- setFMHContained 96
- setHighlight 241
- setInputMessage 167
- setJournalTypeld 148
- setKind 156
- setLanguage 267
- setLine 241
- setNewLine 241
- setNextCommArea 241
- setNextInputMessage 242
- setNextTransId 242
- setPrefix 148
- setPriority 220
- setQueueName 203
- setReplyTimeout 108
- setReturnTermId 203, 204
- setReturnTransId 204
- setRouteCodes 108
- setRouteOption 181
- setStartOpts 204
- setStatus 135
- setTimerECA 82
- setWaitText 220
- signoff 242
- signon 242
- start 204
- startRequestQ 113
- startType 220
- state 198
- stateText 198
- summary 124, 126, 162
- suspend 220
- syncLevel 198

Public methods (*continued*)

- sysId 212
- system 113
- task 114
- terminal 114
- text 162
- time 79, 101
- timeInHours 79, 253
- timeInMinutes 79, 254
- timeInSeconds 79, 254
- timerECA 82
- transId 220
- triggerDataQueueId 220
- tryLock 188
- type 127, 136, 173, 188, 254
- typeText 127
- unload 167
- unlock 188
- unlockRecord 136
- update 101
- userId 221
- value 157
- verifyPassword 267
- wait 148
- waitExternal 221
- waitForAID 243
- waitOnAlarm 221
- width 243
- workArea 212, 222, 243
- write 108
- writeAndGetReply 109
- writeItem 118, 227
- writeRecord 136, 149
- year 79, 101
- purgeable
 - in WaitPurgeability 224
- put
 - in Example of polymorphic behavior 61
 - in lccConsole class 107
 - in lccDataQueue class 118
 - in lccJournal class 148
 - in lccResource class 179
 - in lccSession class 196
 - in lccTempStore class 226
 - in lccTerminal class 237
 - in Polymorphic Behavior 60

Q

- queue
 - in AllocateOpt 199
 - in NextTransIdOpt 245
- queueName
 - in Accessing start data 36
 - in lccStartRequestQ class 202
- queueName (parameter)
 - in Constructor 117, 121
 - in operator= 121
 - in setQueueName 203

R

- rAbendTask
 - in HandleEventReturnOpt 182
- Range
 - in Enumerations 105
 - in lccCondition structure 105
- RBA 29
- rba (parameter)
 - in operator!= 172
 - in operator= 171
 - in operator== 171
- rContinue
 - in HandleEventReturnOpt 182
- readable
 - in Access 137
- reading data 39
- Reading data
 - in Transient Data 39
 - in Using CICS Services 39
- Reading ESDS records
 - in File control 30
 - in Reading records 30
- reading items 41
- Reading items
 - in Temporary storage 41
 - in Using CICS Services 41
- Reading KSDS records
 - in File control 30
 - in Reading records 30
- Reading records
 - in File control 29
 - in Using CICS Services 29
 - Reading ESDS records 30
 - Reading KSDS records 30
 - Reading RRDS records 30
- Reading RRDS records
 - in File control 30
 - in Reading records 30
- readItem
 - in Example of Temporary Storage 43
 - in lccDataQueue class 118
 - in lccTempStore class 226
 - in Reading data 39
 - in Reading items 41
 - in Scope of data in lccBuf reference returned from 'read' methods 63
 - in Temporary storage 41
 - in Transient Data 39
 - in Working with lccResource subclasses 27
- ReadMode
 - in Enumerations 137
 - in lccFile class 137
- readNextItem
 - in lccTempStore class 226
 - in Scope of data in lccBuf reference returned from 'read' methods 63
 - in Temporary storage 41
- readNextRecord
 - in Browsing records 32
 - in lccFileIterator class 141
 - in Public methods 141
- readNextRecord method 32
- READONLY
 - in ASRAStorageType 73
- readPreviousRecord 32
 - in Browsing records 32
 - in lccFileIterator class 142
- readRecord
 - in C++ Exceptions and the Foundation Classes 53
 - in Deleting locked records 32
 - in lccFile class 133
 - in Reading records 30
 - in Updating records 31
- readRecord method 30
- receive
 - in lccSession class 196
 - in lccTerminal class 237
 - in Receiving data from a terminal 44
- receive3270data
 - in Receiving data from a terminal 44
- receive3270Data
 - in lccTerminal class 237
 - in Public methods 237
- receiving data from a terminal 44
- Receiving data from a terminal
 - in Terminal control 44
 - in Using CICS Services 44
- record (parameter)
 - in writeRecord 149
- recordFormat
 - in lccFile class 134
 - in Reading ESDS records 30
 - in Reading RRDS records 30
 - in Writing ESDS records 31
 - in Writing RRDS records 31
- recordFormat method 30
- recordIndex
 - in lccFile class 134
 - in Reading ESDS records 30
 - in Reading KSDS records 30
 - in Reading RRDS records 30
 - in Writing ESDS records 31
 - in Writing KSDS records 31
 - in Writing RRDS records 31
- recordIndex method 30
- recordLength
 - in lccFile class 134
 - in Reading ESDS records 30
 - in Reading KSDS records 30
 - in Reading RRDS records 30
 - in Writing ESDS records 31
 - in Writing KSDS records 31
 - in Writing RRDS records 31
- recordLength method 30
- red
 - in Color 244
- registerData 201
 - in Example of starting transactions 37
 - in lccStartRequestQ class 202
 - in Starting transactions 36
- registerInputMessage 165
 - in lccTerminal class 167

- registerPrefix
 - in IccJournal class 148
 - in Public methods 148
- registerRecordIndex 30
 - in IccFile class 134
 - in Reading ESDS records 30
 - in Reading KSDS records 30
 - in Reading RRDS records 30
 - in Writing ESDS records 31
 - in Writing KSDS records 31
 - in Writing records 30
 - in Writing RRDS records 31
- registerRecordIndex method 30
- relative byte address 29
- relative record number 29
- release
 - in IccSystem class 212
- releaseAtTaskEnd
 - in LoadOpt 168
- releaseText
 - in IccSystem class 212
- remoteTermId
 - in Example of starting transactions 37
- replace
 - in IccBuf class 95
 - in IccBuf constructors 26
- replyTimeout
 - in IccConsole class 107
- req
 - in Example of starting transactions 38
- req1
 - in Example of starting transactions 37
- req2
 - in Example of starting transactions 37
- requestName (parameter)
 - in operator= 175
- reqId (parameter)
 - in cancel 201
 - in cancelAlarm 99
 - in delay 216
 - in setAlarm 101
 - in start 205
- requestName (parameter)
 - in Constructor 175
 - in operator= 82, 175
- requestNum (parameter)
 - in wait 149
- reset
 - in Browsing records 32
 - in IccFileIterator class 142
 - in IccStartRequestQ class 202
- resetAbendHandler
 - in IccControl class 112
- resetRouteCodes
 - in IccConsole class 108
 - in Public methods 108
- resId (parameter)
 - in beginBrowse 209
- resName (parameter)
 - in beginBrowse 209, 210
 - in Constructor 183
- resource (parameter)
 - in beginBrowse 209
 - in Constructor 187
 - in endBrowse 210
 - in enterTrace 217
- resource class 19
- Resource classes
 - in Overview of the foundation classes 19
- resource identification class 18
- Resource identification classes
 - in Overview of the foundation classes 18
- resource object
 - creating 21
- ResourceType
 - in Enumerations 213
 - in IccSystem class 213
- respectAbendHandler
 - in AbendHandlerOpt 222
- retrieveData
 - in Accessing start data 36
 - in IccStartRequestQ class 201, 202
 - in Mapping EXEC CICS calls to Foundation Class methods 285
- RetrieveOpt
 - in Enumerations 206
 - in IccStartRequestQ class 206
- return
 - in Mapping EXEC CICS calls to Foundation Class methods 285
- returnCondition
 - in NoSpaceOpt 228
- returnProgramId
 - in IccControl class 113
 - in Public methods 113
- returnTermId
 - in Accessing start data 36
 - in IccStartRequestQ class 203
- returnToCICS
 - in Functions 68
 - in Icc structure 68
- returnTransId
 - in Accessing start data 36
 - in IccStartRequestQ class 203
- reverse
 - in Highlight 245
- rewriteltem
 - in Example of Temporary Storage 43
 - in IccTempStore class 227
 - in Temporary storage 41
 - in Updating items 42
 - in Writing items 41
- rewriteRecord
 - in IccFile class 135
 - in Updating records 31
- rewriteRecord method 31
- rewriting records 31
- rollBackUOW
 - in IccTask class 219
- routeOption
 - in IccResource class 180

- row (parameter)
 - in send 238
 - in setCursor 241
- RRDS file
 - in File control 29
- RRN 29
- rrn (parameter)
 - in operator!= 186
 - in operator= 185
 - in operator== 185
- rThrowException
 - in HandleEventReturnOpt 182
- run
 - in Base classes 17
 - in C++ Exceptions and the Foundation Classes 52
 - in Example of file control 32, 34
 - in Example of managing transient data 40, 41
 - in Example of polymorphic behavior 60
 - in Example of starting transactions 37
 - in Example of Temporary Storage 42, 43
 - in Example of terminal control 44, 45
 - in Example of time and date services 46
 - in Hello World 10
 - in lccControl class 111, 113
 - in main function 275, 276
 - in Mapping EXEC CICS calls to Foundation Class methods 285
 - in Program control 34
- 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 63
- Scope of data in lccBuf reference returned from 'read' methods
 - in Miscellaneous 63
- scope of references 63
- search (parameter)
 - in Constructor 141
 - in reset 142
- SearchCriterion
 - in Enumerations 138
 - in lccFile class 138
- Second Screen
 - in ICC\$PRG1 (IPR1) 294
 - in Output from sample programs 294
- seconds
 - in lccAbsTime class 79
 - in lccTime class 253
- seconds (parameter)
 - in Constructor 253, 257, 259
 - in set 257, 259

- seconds (parameter) *(continued)*
 - in setReplyTimeout 108
- send
 - in Example of terminal control 44
 - in Hello World 10
 - in lccSession class 196
 - in lccTerminal class 238
 - in Sending data to a terminal 43
- send (parameter)
 - in converse 193
 - in put 107
 - in send 196
 - in sendInvite 197
 - in sendLast 197
 - in write 108, 109
 - in writeAndGetReply 109
- send3270Data
 - in lccTerminal class 239
- sending data to a terminal 43
- Sending data to a terminal
 - in Terminal control 43
 - in Using CICS Services 43
- sendInvite
 - in lccSession class 197
- sendLast
 - in lccSession class 197
- sendLine
 - in Example of file control 33
 - in Example of terminal control 44
 - in lccTerminal class 239, 240
 - in Sending data to a terminal 43
- SendOpt
 - in Enumerations 199
 - in lccSession class 199
- sequential reading of files 32
- session
 - in FacilityType 223
 - in lccControl class 113
- set
 - in lccTimeInterval class 257
 - in lccTimeOfDay class 259
- set (parameter)
 - in boolText 67
- set...
 - in Sending data to a terminal 43
- setAbendHandler
 - in lccControl class 113
- setAccess
 - in lccFile class 135
- setActionOnAnyCondition
 - in lccResource class 180
- setActionOnCondition
 - in lccResource class 180
- setActionsOnConditions
 - in lccResource class 180
- setAlarm
 - in lccAlarmRequestId class 81
 - in lccClock class 101
- setAllRouteCodes
 - in lccConsole class 108

- setClassName
 - in lccBase class 86
 - in Protected methods 86
- setColor
 - in Example of terminal control 45
 - in lccTerminal class 240
- setCursor
 - in lccTerminal class 240, 241
- setCustomClassNum
 - in lccBase class 86
 - in Protected methods 86
- setData 201
 - in lccStartRequestQ class 203
 - in Starting transactions 36
- setDataLength
 - in lccBuf class 96
- setDumpOpts
 - in lccTask class 219
- setEDF
 - in Functions 68
 - in lcc structure 68
 - in lccResource class 180
- setEmptyOnOpen
 - in lccFile class 135
 - in Public methods 135
- setFMHContained
 - in lccBuf class 96
 - in Public methods 96
- setHighlight
 - in Example of terminal control 45
 - in lccTerminal class 241
- setInputMessage 165
 - in lccProgram class 167
 - in Public methods 167
- setJournalTypeld
 - in lccJournal class 148
- setKind
 - in Example of file control 33
 - in lccKey class 156
- setLanguage
 - in lccUser class 267
- setLine
 - in lccTerminal class 241
- setNewLine
 - in lccTerminal class 241
- setNextCommArea
 - in lccTerminal class 241
 - in Public methods 241
- setNextInputMessage
 - in lccTerminal class 242
- setNextTransld
 - in lccTerminal class 242
- setPrefix
 - in lccJournal class 148
- setPriority
 - in lccTask class 220
 - in Public methods 220
- setQueueName
 - in Example of starting transactions 38
 - in lccStartRequestQ class 203
 - in Starting transactions 36
- setReplyTimeout
 - in lccConsole class 108
- setReturnTermld
 - in Example of starting transactions 37
 - in lccStartRequestQ class 203, 204
 - in Starting transactions 36
- setReturnTransld
 - in Example of starting transactions 37
 - in lccStartRequestQ class 204
 - in Starting transactions 36
- setRouteCodes
 - in lccConsole class 108
- setRouteOption
 - in Example of starting transactions 37, 39
 - in lccResource class 181
 - in Program control 35
 - in Public methods 181
- setStartOpts
 - in lccStartRequestQ class 204
- setStatus
 - in lccFile class 135
- setTimerECA
 - in lccAlarmRequestld class 82
- setWaitText
 - in lccTask class 220
- Severe error handling (abendTask)
 - in CICS conditions 56
 - in Conditions, errors, and exceptions 56
- SeverityOpt
 - in Enumerations 110
 - in lccConsole class 110
- signoff
 - in lccTerminal class 242
- signon
 - in lccTerminal class 242
 - in Public methods 242
- singleton class 22
- Singleton classes
 - in Creating a resource object 22
 - in Using CICS resources 22
- size (parameter)
 - in getStorage 211, 217
 - in operator new 86
- start
 - in Example of starting transactions 38
 - in lccRequestld class 175
 - in lccStartRequestQ class 201, 204
 - in Mapping EXEC CICS calls to Foundation Class methods 285
 - in Parameter passing conventions 63
 - in Starting transactions 36
- Starting transactions
 - in Starting transactions asynchronously 36
 - in Using CICS Services 36
- starting transactions asynchronously 36
- Starting transactions asynchronously
 - Accessing start data 36
 - Cancelling unexpired start requests 36
 - Example of starting transactions 36
 - in Using CICS Services 36
 - Starting transactions 36

- startIO
 - in Options 150
- startRequest
 - in StartType 223
- startRequestQ
 - in Example of starting transactions 37, 38
 - in lccControl class 113
- startType
 - in Example of starting transactions 38
 - in lccTask class 220
- StartType
 - in Enumerations 223
 - in lccTask class 223
- state
 - in lccSession class 198
- StateOpt
 - in Enumerations 199
 - in lccSession class 199
- stateText
 - in lccSession class 198
- Status
 - in Enumerations 138
 - in lccFile class 138
- status (parameter)
 - in setStatus 135
- Storage management
 - in Miscellaneous 61
- StorageOpts
 - in Enumerations 224
 - in lccTask class 224
- storageOpts (parameter)
 - in getStorage 211, 217, 218
- storeName (parameter)
 - in Constructor 225
- SUBSPACE
 - in ASRASpaceType 73
- summary
 - in lccEvent class 124
 - in lccException class 126
 - in lccMessage class 162
- support classes 20
- Support Classes
 - in Overview of the foundation classes 20
- suppressDump
 - in AbendDumpOpt 223
- suspend
 - in lccTask class 220
 - in NoSpaceOpt 228
- symbolic debuggers 50
- Symbolic Debuggers
 - in Compiling, executing, and debugging 50
 - in Debugging Programs 50
- synchronous
 - in Options 150
- syncLevel
 - in lccSession class 198
- SyncLevel
 - in Enumerations 200
 - in lccSession class 200
- sysId
 - in lccSystem class 212

- sysId (parameter)
 - in Constructor 191
 - in setRouteOption 181
- sysName (parameter)
 - in Constructor 191
 - in setRouteOption 181
- system
 - in lccControl class 113

T

- task
 - in lccControl class 114
 - in LifeTime 189
- temporary storage
 - deleting items 42
 - example 42
 - introduction 41
 - reading items 41
 - updating items 42
 - Writing items 41
- Temporary storage
 - Deleting items 42
 - Example of Temporary Storage 42
 - in Using CICS Services 41
 - Reading items 41
 - Updating items 42
 - Writing items 41
- termId (parameter)
 - in setReturnTermId 204
 - in start 205
- terminal
 - finding out about 44
 - in FacilityType 223
 - in Hello World 9
 - in lccControl class 114
 - receiving data from 44
 - sending data to 43
- terminal control
 - example 44
 - finding out information 44
 - introduction 43
 - receiving data 44
 - sending data 43
- Terminal control
 - Example of terminal control 44
 - Finding out information about a terminal 44
 - in Using CICS Services 43
 - Receiving data from a terminal 44
 - Sending data to a terminal 43
- terminalInput
 - in StartType 223
- termName (parameter)
 - in setReturnTermId 204
- Test
 - in C++ Exceptions and the Foundation Classes 51, 52
- test (parameter)
 - in boolText 67
- text
 - in lccMessage class 162

- text (parameter)
 - in Constructor 89, 90, 161
 - in operator!= 156
 - in operator<< 94, 236
 - in operator+= 93
 - in operator= 93
 - in operator== 156
 - in writeItem 118, 227
- throw
 - in C++ Exceptions and the Foundation Classes 51
 - in Exception handling (throwException) 55
- throwException
 - in ActionOnCondition 182
 - in CICS conditions 54
- ti
 - in Example of starting transactions 37, 38
- time
 - in lccAbsTime class 79
 - in lccClock class 101
- time (parameter)
 - in Constructor 77, 257, 259
 - in delay 216
 - in setAlarm 101
 - in start 205
- Time and date services
 - Example of time and date services 45
 - in Using CICS Services 45
- time services 45
- timeInHours
 - in lccAbsTime class 79
 - in lccTime class 253
- timeInMinutes
 - in lccAbsTime class 79
 - in lccTime class 254
- timeInSeconds
 - in lccAbsTime class 79
 - in lccTime class 254
- timeInterval
 - in Type 255
- timeInterval (parameter)
 - in operator= 257
- timeOfDay
 - in Type 255
- timeOfDay (parameter)
 - in operator= 259
- timerECA
 - in lccAlarmRequestId class 82
- timerECA (parameter)
 - in Constructor 81
 - in setTimerECA 82
- timeSeparator (parameter)
 - in time 79, 101
- TPName (parameter)
 - in connectProcess 192, 193
- traceNum (parameter)
 - in enterTrace 217
- TraceOpt
 - in Enumerations 224
 - in lccTask class 224
- tracing
 - activating trace output 50
- Tracing a Foundation Class Program
 - Activating the trace output 50
 - in Compiling, executing, and debugging 50
 - in Debugging Programs 50
- transId
 - in lccTask class 220
- transId (parameter)
 - in setNextTransId 242
- transId (parameter)
 - in cancel 201
 - in connectProcess 192
 - in link 166
 - in setNextTransId 242
 - in setReturnTransId 204
 - in start 205
- transient data
 - deleting queues 40
 - example 40
 - introduction 39
 - reading data 39
 - Writing data 40
- Transient Data
 - Deleting queues 40
 - Example of managing transient data 40
 - in Using CICS Services 39
 - Reading data 39
 - Writing data 40
- transName (parameter)
 - in setReturnTransId 204
- triggerDataQueueId
 - in lccTask class 220
- trueFalse (parameter)
 - in setEmptyOnOpen 135
- try
 - in C++ Exceptions and the Foundation Classes 51, 52
 - in Exception handling (throwException) 55, 56
 - in main function 276
- tryLock
 - in lccSemaphore class 188
- tryNumber
 - in C++ Exceptions and the Foundation Classes 51, 52
- type
 - in C++ Exceptions and the Foundation Classes 52
 - in lccException class 127
 - in lccFile class 136
 - in lccRecordIndex class 173
 - in lccSemaphore class 188
 - in lccTime class 254
- Type
 - in Enumerations 127, 174, 255
 - in lccException class 127
 - in lccRecordIndex class 174
 - in lccTime class 255
- type (parameter)
 - in condition 123, 178
 - in Constructor 85, 89, 173, 183, 187
 - in waitExternal 221
- typeText
 - in lccException class 127

U

- underscore
 - in Highlight 245
- UNIX
 - in ClassMemoryMgmt 69
 - in Storage management 62
- unknownException
 - in Functions 69
 - in lcc structure 69
- unload
 - in lccProgram class 167
- unlock
 - in lccSemaphore class 188
- unlockRecord
 - in lccFile class 136
- UOW
 - in LifeTime 189
- updatable
 - in Access 137
- update
 - in lccClock class 101
 - in ReadMode 137
- update (parameter)
 - in Constructor 99
- UpdateMode
 - in Enumerations 103
 - in lccClock class 103
- updateToken (parameter)
 - in deleteLockedRecord 130
 - in readNextRecord 141
 - in readPreviousRecord 142
 - in readRecord 133
 - in rewriteRecord 135
 - in unlockRecord 136
- updating items 42
- Updating items
 - in Temporary storage 42
 - in Using CICS Services 42
- updating records 31
- Updating records
 - in File control 31
 - in Using CICS Services 31
- upper
 - in Case 244
- USER
 - in ASRAStorageType 73
- user (parameter)
 - in signon 243
- userDataKey
 - in StorageOpts 224
- USEREXECKEY
 - in ASRAKeyType 72
- userId
 - in lccTask class 221
- userId (parameter)
 - in start 205
- userName (parameter)
 - in Constructor 265
- Using an object
 - in C++ Objects 16
- using CICS resources 21

- Using CICS resources
 - Calling methods on a resource object 22
 - Creating a resource object 21
 - in Overview of the foundation classes 21
 - Singleton classes 22
- Using CICS Services
 - Accessing start data 36
 - Browsing records 32
 - Cancelling unexpired start requests 36
 - Deleting items 42
 - Deleting queues 40
 - Deleting records 31
 - Example of file control 32
 - Example of managing transient data 40
 - Example of starting transactions 36
 - Example of Temporary Storage 42
 - Example of terminal control 44
 - Example of time and date services 45
 - Finding out information about a terminal 44
 - Reading data 39
 - Reading items 41
 - Reading records 29
 - Receiving data from a terminal 44
 - Sending data to a terminal 43
 - Starting transactions 36
 - Updating items 42
 - Updating records 31
 - Writing data 40
 - Writing items 41
 - Writing records 30

V

- value
 - in lccKey class 157
- value (parameter)
 - in operator= 156
- variable (parameter)
 - in Foundation Classes—reference 66
- verifyPassword
 - in lccUser class 267
 - in Public methods 267
- virtual
 - in Glossary 299
- VSAM 29

W

- wait
 - in lccJournal class 148
 - in SendOpt 199
- waitExternal
 - in lccTask class 221
- waitForAID
 - in Example of terminal control 45
 - in lccTerminal class 243
- waitOnAlarm
 - in lccAlarmRequestId class 81
 - in lccTask class 221
- WaitPostType
 - in Enumerations 224

- WaitPostType (*continued*)
 - in lccTask class 224
- WaitPurgeability
 - in Enumerations 224
 - in lccTask class 224
- width
 - in lccTerminal class 243
- workArea
 - in lccSystem class 212
 - in lccTask class 222
 - in lccTerminal class 243
- Working with lccResource subclasses
 - in Buffer objects 26
 - in lccBuf class 26
- write
 - in lccConsole class 108
- writeAndGetReply
 - in lccConsole class 109
- writelnItem
 - in C++ Exceptions and the Foundation Classes 53
 - in Calling methods on a resource object 23
 - in lccDataQueue class 118
 - in lccTempStore class 227
 - in Temporary storage 41
 - in Transient Data 39
 - in Working with lccResource subclasses 27
 - in Writing data 40
 - in Writing items 41
- writeRecord
 - in Example of file control 33
 - in lccFile class 136
 - in lccJournal class 149
 - in Writing KSDS records 31
 - in Writing records 30
 - in Writing RRDS records 31
- writeRecord method
 - lccFile class 30
- Writing data 40
 - in Transient Data 40
 - in Using CICS Services 40
- Writing ESDS records
 - in File control 31
 - in Writing records 31
- Writing items 41
 - in Temporary storage 41
 - in Using CICS Services 41
- Writing KSDS records
 - in File control 31
 - in Writing records 31
- Writing records
 - in File control 30
 - in Using CICS Services 30
 - Writing ESDS records 31
 - Writing KSDS records 31
 - Writing RRDS records 31
- Writing RRDS records
 - in File control 31
 - in Writing records 31

X

- X
 - in actionOnConditionAsChar 177
 - in operatingSystem 211
- xldb 50
- XPLINK 6

Y

- year
 - in lccAbsTime class 79
 - in lccClock class 101
- yellow
 - in Color 244
- yesNo (parameter)
 - in setFMHCContained 96

Notices

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

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

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

Other product and service names might be trademarks of IBM or other companies.

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:

IBM United Kingdom Limited
User Technologies Department (MP095)
Hursley Park
Winchester
Hampshire
SO21 2JN
United Kingdom

- By fax:
 - From outside the U.K., after your international access code use 44-1962-816151
 - From within the U.K., use 01962-816151
- Electronically, use the appropriate network ID:
 - 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.



Product Number: 5655-M15

SC34-6437-03



Spine information:



CICS TS for z/OS

C++ OO Class Libraries

Version 3
Release 1