

CICS Transaction Server for z/OS



C++ OO Class Libraries

Version 3 Release 2

CICS Transaction Server for z/OS



C++ OO Class Libraries

Version 3 Release 2

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 355.

This edition applies to Version 3 Release 2 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.

© 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
Softcopy links	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	7
Chapter 3. Hello World	9
Compile and link "Hello World"	10
Running "Hello World" on your CICS server	10
Expected Output from "Hello World"	10

Part 2. Using the CICS foundation classes 11

Chapter 4. C++ Objects	13
Creating an object.	13
Using an object.	14
Deleting an object.	14
Chapter 5. Overview of the foundation classes	15
Base classes	15
Resource identification classes	16
Resource classes	17
Support Classes	18
Using CICS resources	19
Creating a resource object	19
Calling methods on a resource object	21
Chapter 6. Buffer objects	23
IccBuf class	23
Data area ownership.	23
Data area extensibility	23
IccBuf constructors	23
IccBuf methods.	24
Working with IccResource subclasses	25
Chapter 7. Using CICS Services	27
File control	27

Reading records	27
Writing records	28
Updating records	29
Deleting records	30
Browsing records	30
Example of file control	30
Program control	32
Starting transactions asynchronously	34
Starting transactions	34
Accessing start data	34
Cancelling unexpired start requests	34
Example of starting transactions	35
Transient Data	37
Reading data	38
Writing data	38
Deleting queues	38
Example of managing transient data	38
Temporary storage	39
Reading items	40
Writing items	40
Updating items	40
Deleting items	40
Example of Temporary Storage	40
Terminal control	42
Sending data to a terminal	42
Receiving data from a terminal	42
Finding out information about a terminal	42
Example of terminal control	42
Time and date services	44
Example of time and date services	44
Chapter 8. Compiling, executing, and debugging	47
Compiling Programs	47
Executing Programs	47
Debugging Programs	47
Symbolic Debuggers	48
Tracing a Foundation Class Program	48
Execution Diagnostic Facility	48
Chapter 9. Conditions, errors, and exceptions	49
Foundation Class Abend codes	49
C++ Exceptions and the Foundation Classes	49
CICS conditions	51
Manual condition handling (noAction)	52
Automatic condition handling (callHandleEvent)	52
Exception handling (throwException)	53
Severe error handling (abendTask)	54
Platform differences	54
Object level	54
Method level	55
Parameter level	55
Chapter 10. Miscellaneous	57
Polymorphic Behavior	57
Example of polymorphic behavior	58
Storage management	59

Parameter passing conventions	60
Scope of data in lccBuf reference returned from 'read' methods	61

Part 3. Foundation Classes—reference 63

Chapter 11. lcc structure	65
Functions	65
boolText	65
catchException	65
conditionText.	65
initializeEnvironment	66
isClassMemoryMgmtOn	66
isEDFOn	66
isFamilySubsetEnforcementOn	66
returnToCICS	66
setEDF.	67
unknownException	67
Enumerations	67
Bool	67
BoolSet	67
ClassMemoryMgmt	68
FamilySubset	68
GetOpt	68
Platforms	69
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	75
Inherited protected methods	75
Chapter 13. lccAbsTime class	77
lccAbsTime constructor	77
Constructor (1)	77
Constructor (2)	77
Public methods	77
date	77
dayOfMonth	78
dayOfWeek	78
daysSince1900	78
hours	78
milliSeconds	78
minutes	78
monthOfYear.	79

operator=	79
packedDecimal	79
seconds	79
time	79
timeInHours	79
timeInMinutes	80
timeInSeconds	80
year	80
Inherited public methods	80
Inherited protected methods	81
Chapter 14. IccAlarmRequestId class	83
IccAlarmRequestId constructors	83
Constructor (1)	83
Constructor (2)	83
Constructor (3)	83
Public methods	83
isExpired	83
operator= (1)	84
operator= (2)	84
operator= (3)	84
setTimerECA	84
timerECA	84
Inherited public methods	84
Inherited protected methods	85
Chapter 15. IccBase class	87
IccBase constructor (protected)	87
Constructor	87
Public methods	87
classType	87
className	87
customClassNum	88
operator delete	88
operator new	88
Protected methods	88
setClassName	88
setCustomClassNum	88
Enumerations	89
ClassType	89
NameOpt	89
Chapter 16. IccBuf class	91
IccBuf constructors	91
Constructor (1)	91
Constructor (2)	91
Constructor (3)	91
Constructor (4)	92
Public methods	92
append (1)	92
append (2)	92
assign (1)	92
assign (2)	93
cut	93
dataArea	93
dataAreaLength	93

dataAreaOwner	93
dataAreaType	94
dataLength	94
insert	94
isFMHContained	94
operator const char*	94
operator= (1)	95
operator= (2)	95
operator+= (1)	95
operator+= (2)	95
operator==	95
operator!=.	96
operator« (1).	96
operator« (2).	96
operator« (3).	96
operator« (4).	96
operator« (5).	96
operator« (6).	96
operator« (7).	97
operator« (8).	97
operator« (9).	97
operator« (10)	97
operator« (11)	97
operator« (12)	97
operator« (13)	97
operator« (14)	97
operator« (15)	98
overlay	98
replace	98
setDataLength	98
setFMHContained	99
Inherited public methods	99
Inherited protected methods	99
Enumerations	99
DataAreaOwner	99
DataAreaType	99
Chapter 17. IccClock class	101
IccClock constructor	101
Constructor	101
Public methods	101
absTime	101
cancelAlarm	101
date	102
dayOfMonth	102
dayOfWeek.	102
daysSince1900	102
milliSeconds	102
monthOfYear	103
setAlarm	103
time	103
update	104
year	104
Inherited public methods	104
Inherited protected methods	104
Enumerations	104

DateFormat	104
DayOfWeek	105
MonthOfYear	105
UpdateMode	105
Chapter 18. IccCondition structure	107
Enumerations	107
Codes.	107
Range	107
Chapter 19. IccConsole class	109
IccConsole constructor (protected)	109
Constructor	109
Public methods	109
instance	109
put	109
replyTimeout	109
resetRouteCodes	110
setAllRouteCodes	110
setReplyTimeout (1)	110
setReplyTimeout (2)	110
setRouteCodes	110
write	110
writeAndGetReply	111
Inherited public methods	111
Inherited protected methods.	112
Enumerations	112
SeverityOpt.	112
Chapter 20. IccControl class	113
IccControl constructor (protected).	113
Constructor	113
Public methods	113
callingProgramId	113
cancelAbendHandler	113
commArea	113
console	114
initData	114
instance	114
isCreated	114
programId	114
resetAbendHandler	115
returnProgramId	115
run	115
session	115
setAbendHandler (1)	115
setAbendHandler (2)	115
startRequestQ.	116
system	116
task	116
terminal	116
Inherited public methods	116
Inherited protected methods.	117
Chapter 21. IccConvId class	119
IccConvId constructors	119

Constructor (1)	119
Constructor (2)	119
Public methods	119
operator= (1)	119
operator= (2)	119
Inherited public methods	119
Inherited protected methods	120
Chapter 22. IccDataQueue class	121
IccDataQueue constructors	121
Constructor (1)	121
Constructor (2)	121
Public methods	121
clear	121
empty	121
get	122
put	122
readItem	122
writeItem (1)	122
writeItem (2)	122
Inherited public methods	123
Inherited protected methods	123
Chapter 23. IccDataQueueeld class	125
IccDataQueueeld constructors	125
Constructor (1)	125
Constructor (2)	125
Public methods	125
operator= (1)	125
operator= (2)	125
Inherited public methods	125
Inherited protected methods	126
Chapter 24. IccEvent class	127
IccEvent constructor	127
Constructor	127
Public methods	127
className	127
classType	127
condition	127
conditionText	128
methodName	128
summary	128
Inherited public methods	128
Inherited protected methods	128
Chapter 25. IccException class	129
IccException constructor	129
Constructor	129
Public methods	130
className	130
classType	130
message	130
methodName	130
number	130
summary	130

type	131
typeText	131
Inherited public methods	131
Inherited protected methods	131
Enumerations	131
Type	131
Chapter 26. IccFile class	133
IccFile constructors	133
Constructor (1)	133
Constructor (2)	133
Public methods	133
access	133
accessMethod	134
beginInsert(VSAM only)	134
deleteLockedRecord	134
deleteRecord	134
enableStatus	135
endInsert(VSAM only)	135
isAddable	135
isBrowsable	136
isDeletable	136
isEmptyOnOpen	136
isReadable	136
isRecoverable	137
isUpdatable	137
keyLength	137
keyPosition	137
openStatus	138
readRecord	138
recordFormat	138
recordIndex	139
recordLength	139
registerRecordIndex	139
rewriteRecord	139
setAccess	140
setEmptyOnOpen	140
setStatus	140
type	140
unlockRecord	141
writeRecord	141
Inherited public methods	142
Inherited protected methods	142
Enumerations	142
Access	142
ReadMode	143
SearchCriterion	143
Status	143
Chapter 27. IccFileId class	145
IccFileId 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 28. IccFileIterator class	147
IccFileIterator constructor	147
Constructor	147
Public methods	147
readNextRecord	147
readPreviousRecord	148
reset	148
Inherited public methods	148
Inherited protected methods	149
Chapter 29. IccGroupId class	151
IccGroupId constructors	151
Constructor (1)	151
Constructor (2)	151
Public methods	151
operator= (1)	151
operator= (2)	151
Inherited public methods	151
Inherited protected methods	152
Chapter 30. IccJournal class	153
IccJournal constructors	153
Constructor (1)	153
Constructor (2)	153
Public methods	153
clearPrefix	153
journalTypeId	154
put	154
registerPrefix	154
setJournalTypeId (1)	154
setJournalTypeId (2)	154
setPrefix (1)	154
setPrefix (2)	154
wait	155
writeRecord (1)	155
writeRecord (2)	155
Inherited public methods	155
Inherited protected methods	156
Enumerations	156
Options	156
Chapter 31. IccJournalId class	157
IccJournalId constructors	157
Constructor (1)	157
Constructor (2)	157
Public methods	157
number	157
operator= (1)	157
operator= (2)	157
Inherited public methods	158
Inherited protected methods	158
Chapter 32. IccJournalTypeId class	159

lccJournalTypeId 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 33. lccKey class	161
lccKey constructors	161
Constructor (1)	161
Constructor (2)	161
Constructor (3)	161
Public methods	161
assign	161
completeLength	161
kind	162
operator= (1)	162
operator= (2)	162
operator= (3)	162
operator== (1)	162
operator== (2)	162
operator== (3)	162
operator!= (1)	162
operator!= (2)	162
operator!= (3)	163
setKind	163
value	163
Inherited public methods	163
Inherited protected methods	163
Enumerations	163
Kind	163
Chapter 34. lccLockId class	165
lccLockId constructors	165
Constructor (1)	165
Constructor (2)	165
Public methods	165
operator= (1)	165
operator= (2)	165
Inherited public methods	165
Inherited protected methods	166
Chapter 35. lccMessage class	167
lccMessage constructor	167
Constructor	167
Public methods	167
className	167
methodName	167
number	167
summary	168
text	168
Inherited public methods	168
Inherited protected methods	168

Chapter 36. IccPartnerId class	169
IccPartnerId 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 37. IccProgram class	171
IccProgram constructors	171
Constructor (1)	171
Constructor (2)	171
Public methods	171
address	171
clearInputMessage	171
entryPoint	172
length	172
link	172
load	173
registerInputMessage	173
setInputMessage	173
unload	173
Inherited public methods	173
Inherited protected methods	174
Enumerations	174
CommitOpt	174
LoadOpt	174
Chapter 38. IccProgramId class	175
IccProgramId constructors	175
Constructor (1)	175
Constructor (2)	175
Public methods	175
operator= (1)	175
operator= (2)	175
Inherited public methods	175
Inherited protected methods	176
Chapter 39. IccRBA class	177
IccRBA constructor	177
Constructor	177
Public methods	177
operator= (1)	177
operator= (2)	177
operator== (1)	177
operator== (2)	177
operator!= (1)	178
operator!= (2)	178
number	178
Inherited public methods	178
Inherited protected methods	178
Chapter 40. IccRecordIndex class	179
IccRecordIndex constructor (protected)	179

Constructor	179
Public methods	179
length	179
type	179
Inherited public methods	179
Inherited protected methods	180
Enumerations	180
Type	180
Chapter 41. IccRequestId class	181
IccRequestId constructors	181
Constructor (1)	181
Constructor (2)	181
Constructor (3)	181
Public methods	181
operator= (1)	181
operator= (2)	181
Inherited public methods	182
Inherited protected methods	182
Chapter 42. IccResource class	183
IccResource constructor (protected)	183
Constructor	183
Public methods	183
actionOnCondition	183
actionOnConditionAsChar	183
actionsOnConditionsText	184
clear	184
condition	184
conditionText	185
get	185
handleEvent	185
id	185
isEDFOn	185
isRouteOptionOn	185
name	186
put	186
routeOption	186
setActionOnAnyCondition	186
setActionOnCondition	186
setActionsOnConditions	187
setEDF	187
setRouteOption (1)	187
setRouteOption (2)	187
Inherited public methods	188
Inherited protected methods	188
Enumerations	188
ActionOnCondition	188
HandleEventReturnOpt	188
ConditionType	189
Chapter 43. IccResourceId class	191
IccResourceId constructors (protected)	191
Constructor (1)	191
Constructor (2)	191
Public methods	191

name	191
nameLength	191
Protected methods	192
operator=	192
Inherited public methods	192
Inherited protected methods	192
Chapter 44. lccRRN class	193
lccRRN constructors	193
Constructor	193
Public methods	193
operator= (1)	193
operator= (2)	193
operator== (1)	193
operator== (2)	193
operator!= (1)	194
operator!= (2)	194
number	194
Inherited public methods	194
Inherited protected methods	194
Chapter 45. lccSemaphore class	195
lccSemaphore constructor	195
Constructor (1)	195
Constructor (2)	195
Public methods	195
lifeTime	195
lock	196
tryLock	196
type	196
unlock.	196
Inherited public methods	196
Inherited protected methods	197
Enumerations	197
LockType	197
LifeTime	197
Chapter 46. lccSession class	199
lccSession constructors (public)	199
Constructor (1)	199
Constructor (2)	199
Constructor (3)	199
lccSession constructor (protected)	199
Constructor	199
Public methods	200
allocate	200
connectProcess (1)	200
connectProcess (2)	200
connectProcess (3)	201
converse.	201
convId	201
errorCode	201
extractProcess	202
flush	202
free	202
get	202

isErrorSet	202
isNoDataSet	202
isSignalSet	203
issueAbend	203
issueConfirmation	203
issueError	203
issuePrepare	203
issueSignal	204
PIPList	204
process	204
put	204
receive	204
send (1)	205
send (2)	205
sendInvite (1)	205
sendInvite (2)	205
sendLast (1)	206
sendLast (2)	206
state	206
stateText	207
syncLevel	207
Inherited public methods	207
Inherited protected methods	208
Enumerations	208
AllocateOpt	208
SendOpt	208
StateOpt	208
SyncLevel	208
Chapter 47. IccStartRequestQ class	209
IccStartRequestQ constructor (protected)	209
Constructor	209
Public methods	209
cancel	209
clearData	209
data	210
instance	210
queueName	210
registerData	210
reset	210
retrieveData	211
returnTermId	211
returnTransId	211
setData	211
setQueueName	211
setReturnTermId (1)	212
setReturnTermId (2)	212
setReturnTransId (1)	212
setReturnTransId (2)	212
setStartOpts	212
start	213
Inherited public methods	213
Inherited protected methods	214
Enumerations	214
RetrieveOpt	214
ProtectOpt	214

CheckOpt	214
Chapter 48. IccSysId class	215
IccSysId constructors	215
Constructor (1)	215
Constructor (2)	215
Public methods	215
operator= (1)	215
operator= (2)	215
Inherited public methods	215
Inherited protected methods	216
Chapter 49. IccSystem class	217
IccSystem constructor (protected)	217
Constructor	217
Public methods	217
appName	217
beginBrowse (1)	217
beginBrowse (2)	218
dateFormat	218
endBrowse	218
freeStorage	218
getFile (1)	218
getFile (2)	219
getNextFile	219
getStorage	219
instance	219
operatingSystem	220
operatingSystemLevel	220
release	220
releaseText	220
sysId	221
workArea	221
Inherited public methods	221
Inherited protected methods	221
Enumerations	222
ResourceType	222
Chapter 50. IccTask class	223
IccTask Constructor (protected)	223
Constructor	223
Public methods	223
abend	223
abendData	223
commitUOW	224
delay	224
dump	224
enterTrace	225
facilityType	225
freeStorage	225
getStorage	226
instance	226
isCommandSecurityOn	226
isCommitSupported	226
isResourceSecurityOn	227
isRestarted	227

isStartDataAvailable	227
number	227
principalSysId	227
priority	228
rollBackUOW	228
setDumpOpts	228
setPriority	228
setWaitText	228
startType	229
suspend	229
transId	229
triggerDataQueueId	229
userId	229
waitExternal	230
waitOnAlarm	230
workArea	230
Inherited public methods	231
Inherited protected methods	231
Enumerations	231
AbendHandlerOpt	231
AbendDumpOpt	231
DumpOpts	231
FacilityType	232
StartType	232
StorageOpts	232
TraceOpt	233
WaitPostType	233
WaitPurgeability	233
Chapter 51. IccTempStore class	235
IccTempStore constructors	235
Constructor (1)	235
Constructor (2)	235
Public methods	235
clear	235
empty	236
get	236
numberOfItems	236
put	236
readItem	236
readNextItem	237
rewriteItem	237
writeItem (1)	237
writeItem (2)	238
Inherited public methods	238
Inherited protected methods	238
Enumerations	239
Location	239
NoSpaceOpt	239
Chapter 52. IccTempStoreId class	241
IccTempStoreId constructors	241
Constructor (1)	241
Constructor (2)	241
Public methods	241
operator= (1)	241

operator= (2)	241
Inherited public methods	241
Inherited protected methods	242
Chapter 53. lccTermId class	243
lccTermId constructors	243
Constructor (1)	243
Constructor (2)	243
Public methods	243
operator= (1)	243
operator= (2)	243
Inherited public methods	243
Inherited protected methods	244
Chapter 54. lccTerminal class	245
lccTerminal constructor (protected)	245
Constructor	245
Public methods	245
AID	245
clear	245
cursor	245
data	246
erase	246
freeKeyboard	246
get	246
height	246
inputCursor	246
instance	247
line	247
netName	247
operator« (1)	247
operator« (2)	247
operator« (3)	247
operator« (4)	247
operator« (5)	248
operator« (6)	248
operator« (7)	248
operator« (8)	248
operator« (9)	248
operator« (10)	248
operator« (11)	248
operator« (12)	248
operator« (13)	249
operator« (14)	249
operator« (15)	249
operator« (16)	249
operator« (17)	249
operator« (18)	249
put	249
receive	250
receive3270Data	250
send (1)	250
send (2)	250
send (3)	250
send (4)	251
send3270Data (1)	251

send3270Data (2)	251
send3270Data (3)	252
send3270Data (4)	252
sendLine (1)	252
sendLine (2)	252
sendLine (3)	253
sendLine (4)	253
setColor	253
setCursor (1)	253
setCursor (2)	254
setHighlight	254
setLine	254
setNewLine	254
setNextCommArea	255
setNextInputMessage	255
setNextTransId	255
signoff	255
signon (1)	255
signon (2)	256
waitForAID (1)	256
waitForAID (2)	256
width	257
workArea	257
Inherited public methods	257
Inherited protected methods	257
Enumerations	257
AIDVal	257
Case	258
Color	258
Highlight	258
NextTransIdOpt	258
Chapter 55. IccTerminalData class	259
IccTerminalData constructor (protected)	259
Constructor	259
Public methods	259
alternateHeight	259
alternateWidth	259
defaultHeight	260
defaultWidth	260
graphicCharCodeSet	260
graphicCharSetId	260
isAPLKeyboard	260
isAPLText	261
isBTrans	261
isColor	261
isEWA	261
isExtended3270	261
isFieldOutline	262
isGoodMorning	262
isHighlight	262
isKatakana	262
isMSRControl	262
isPS	263
isSOSI	263
isTextKeyboard	263

isTextPrint	263
isValidation	263
Inherited public methods	264
Inherited protected methods	264
Chapter 56. lccTime class	265
lccTime constructor (protected)	265
Constructor	265
Public methods	265
hours	265
minutes	265
seconds	265
timeInHours	266
timeInMinutes	266
timeInSeconds	266
type	266
Inherited public methods	266
Inherited protected methods	266
Enumerations	267
Type	267
Chapter 57. lccTimeInterval class	269
lccTimeInterval constructors	269
Constructor (1)	269
Constructor (2)	269
Public methods	269
operator=	269
set	269
Inherited public methods	270
Inherited protected methods	270
Chapter 58. lccTimeOfDay class	271
lccTimeOfDay constructors	271
Constructor (1)	271
Constructor (2)	271
Public methods	271
operator=	271
set	271
Inherited public methods	272
Inherited protected methods	272
Chapter 59. lccTPNameId class	273
lccTPNameId constructors	273
Constructor (1)	273
Constructor (2)	273
Public methods	273
operator= (1)	273
operator= (2)	273
Inherited public methods	273
Inherited protected methods	274
Chapter 60. lccTransId class	275
lccTransId constructors	275
Constructor (1)	275
Constructor (2)	275
Public methods	275

operator= (1)	275
operator= (2)	275
Inherited public methods	275
Inherited protected methods	276
Chapter 61. IccUser class	277
IccUser constructors	277
Constructor (1)	277
Constructor (2)	277
Public methods	277
changePassword.	277
daysUntilPasswordExpires	278
ESMReason	278
ESMResponse	278
groupId	278
invalidPasswordAttempts	278
language	278
lastPasswordChange	278
lastUseTime	279
passwordExpiration	279
setLanguage	279
verifyPassword	279
Inherited public methods	279
Inherited protected methods	280
Chapter 62. IccUserId class	281
IccUserId constructors	281
Constructor (1)	281
Constructor (2)	281
Public methods	281
operator= (1)	281
operator= (2)	281
Inherited public methods	281
Inherited protected methods	282
Chapter 63. IccValue structure	283
Enumeration	283
CVDA.	283
Chapter 64. main function	287

Part 4. Appendixes 289

Appendix A. Mapping EXEC CICS calls to Foundation Class methods	291
Appendix B. Mapping Foundation Class methods to EXEC CICS calls	297
Appendix C. Output from sample programs	303
ICC\$BUF (IBUF)	303
ICC\$CLK (ICLK)	303
ICC\$DAT (IDAT)	303
ICC\$EXC1 (IEX1)	304
ICC\$EXC2 (IEX2)	304
ICC\$EXC3 (IEX3)	304
ICC\$FIL (IFIL)	304
ICC\$HEL (IHEL)	304

ICC\$JRN (IJRN)	305
ICC\$PRG1 (IPR1)	305
First Screen	305
Second Screen	305
ICC\$RES1 (IRS1)	305
ICC\$RES2 (IRS2)	306
ICC\$SEM (ISEM)	306
ICC\$SES1 (ISE1)	306
ICC\$SES2 (ISE2)	307
ICC\$SRQ1 (ISR1)	307
ICC\$SRQ2 (ISR2)	307
ICC\$SYS (ISYS)	308
ICC\$TMP (ITMP)	308
ICC\$TRM (ITRM)	308
ICC\$TSK (ITSK)	309
Bibliography	311
The CICS Transaction Server for z/OS library	311
The entitlement set	311
PDF-only books	311
Other CICS books	313
Related books	313
C++ Programming	313
CICS client manuals	313
Determining if a publication is current	314
Accessibility	315
Index	317
Notices	355
Trademarks	357

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:

- Installation and setup describes how to install the product and check that the installation is complete.
- Using the CICS foundation classes describes the classes and how to use them.
- Foundation Classes—reference contains the reference material: the class descriptions and their methods.
- For those of you familiar with the EXEC CICS calls, Appendix A, “Mapping EXEC CICS calls to Foundation Class methods,” on page 291 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 297 maps them the other way — foundation class methods to EXEC CICS calls.
- Appendix C, “Output from sample programs,” on page 303 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 2.

“RACF” is used throughout this book to mean the 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.

Softcopy links

This book is linked to the *CICS Transaction Server for z/OS Glossary* and to the *CICS Messages and Codes* manual.

If you are using BookManager[®] READ/MVS or BookManager READ/VM, you can view the definitions of terms and messages directly from this book by selecting a term with your cursor, and pressing the ENTER key.

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.

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
ICCCCTLEH	IccControl
ICCDATEH	IccDataQueue
ICCEH	see 1 on page 6
ICCEVTEH	IccEvent
ICCEXCEH	IccException
ICCFILEH	IccFile
ICCFLIEH	IccFileIterator
ICCGLBEH	Icc (struct) (global functions)
ICCJKRNEH	IccJournal
ICCMSGEH	IccMessage
ICCPRGEH	IccProgram
ICCRECEH	IccRecordIndex, IccKey, IccRBA and IccRRN
ICCRESEH	IccResource
ICCRIDEH	IccResourceId + subclasses (such as IccConvId)
ICCSEMEH	IccSemaphore
ICCSESEH	IccSession
ICCSRQEH	IccStartRequestQ
ICCSYSEH	IccSystem
ICCTIMEH	IccTime, IccAbsTime, IccTimeInterval, IccTimeOfDay
ICCTMDEH	IccTerminalData
ICCTMPEH	IccTempStore
ICCTRMEH	IccTerminal
ICCTSKEH	IccTask
ICCUSREH	IccUser
ICCVALEH	IccValue (struct)

Note:

1. A single header that #includes all the above header files is supplied as ICCEH
2. The file ICCMAIN is also supplied with the C++ header files. This contains the **main** function stub that should be used when you build a Foundation Class program.

Location

PDS: CICSTS32.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: CICSTS32.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: CICSTS32.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 CICSTS32.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 CICSTS32.CICS.SDFHPROC.

ICCFCCCL 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 47.

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

CICSTS32.CICS.SDFHSDCK contains the member

- ICCFCIMP - 'sidedeck' containing import control statements

CICSTS32.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

CICSTS32.CICS.SDFHLOAD contains the members

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

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 sectionchapter shows how to get started in CICS OO programming. It is intended as an appetizer; Chapter 5, “Overview of the foundation classes,” on page 15 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 "iccmmain.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 287 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 113). 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.

```
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 47 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.

Chapter 4. C++ Objects

This sectionchapter 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, 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 59.

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

- “Base classes”
- “Resource identification classes” on page 16
- “Resource classes” on page 17
- “Support Classes” on page 18.

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

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

Base classes

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 17.

iccResourceId

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

lccTime

The base class for the classes that store time information: **lccAbsTime**, **lccTimeInterval** and **lccTimeOfDay**.

Resource identification classes

lccBase

- lccResourceId
- lccConvId
- lccDataQueueId
- lccFileId
- lccGroupId
- lccJournalId
- lccJournalTypeId
- lccLockId
- lccPartnerId
- lccProgramId
- lccRequestId
 - lccAlarmRequestId
- lccSysId
- lccTempStoreId
- lccTermId
- lccTPNameId
- lccTransId
- lccUserId

Figure 2. Resource identification classes

CICS resource identification classes define CICS resource identifiers – typically entries in one of the CICS tables. For example an **lccFileId** 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 **lccResourceId** class.
- The constructors check that any supplied table entry meets CICS standards. For example, an **lccFileId** 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 **lccFileId** object as a parameter do not accept an **lccProgramId** object instead. If character strings representing the resource names are used instead, the compiler cannot check for validity – it cannot check whether the string is a file name or a program name.

Many of the resource classes, described in “Resource classes” on page 17, contain resource identification classes. For example, an **lccFile** object contains an **lccFileId** object. You must use the resource object, not the resource identification object to operate on a CICS resource. For example, you must use **lccFile**, rather than **lccFileId** to read a record from a file.

Class	CICS resource	CICS table
lccAlarmRequestId	alarm request	
lccConvId	conversation	

Class	CICS resource	CICS table
lccDataQueueId	data queue	
lccFileId	file	FCT
lccGroupId	group	
lccJournalId	journal	
lccJournalTypeId	journal type	
lccLockId	(Not applicable)	
lccPartnerId	APPC partner definition files	
lccProgramId	program	PPT
lccRequestId	request	
lccSysId	remote system	
lccTempStoreId	temporary storage	TST
lccTermId	terminal	TCT
lccTPNameId	remote APPC TP name	
lccTransId	transaction	PCT
lccUserId	user	

Resource classes

lccBase

lccResource

- lccAbendData
- lccClock
- lccConsole
- lccControl
- lccDataQueue
- lccFile
- lccFileIterator
- lccJournal
- lccProgram
- lccSemaphore
- lccSession
- lccStartRequestQ
- lccSystem
- lccTask
- lccTempStore
- lccTerminal
- lccTerminalData
- lccUser

Figure 3. Resource classes

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

- Terminals are modelled by **lccTerminal**.
- Programs are modelled by **lccProgram**.
- Temporary Storage queues are modelled by **lccTempStore**.
- Transient Data queues are modelled by **lccDataQueue**.

All CICS resource classes inherit from the **lccResource** base class. For example, any operation on a CICS resource may raise a CICS condition; the **condition** method of **lccResource** (see page “condition” on page 184) can interrogate it.

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

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

Support Classes

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 23).

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

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

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

Using CICS resources

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

Creating a resource object

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

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

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 task's terminal, provided that your principal facility is a 3270 terminal.

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

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

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


Calling methods on a resource object

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

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

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

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);
```

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

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

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

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

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

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

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

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

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

IccBuf methods

An **IccBuf** object can be manipulated using a number of supplied methods; for example you can append data to the buffer, change the data in the buffer, cut data out of the buffer, or insert data into the middle of the buffer. The operators **const char***, **=**, **+=**, **==**, **!=**, and **<<** have been overloaded in class **IccBuf**. There are also methods that allow the **IccBuf** attributes to be queried. For more details see the reference section.

Working with `IccResource` subclasses

To illustrate this, consider writing a queue item to CICS temporary storage using `IccTempstore` class.

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

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

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

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

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

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

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

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

The C++ compiler 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 chevron <<` method, overloaded in class `IccBuf`). The `writeltem` 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 sectionchapters, which describe how to use the foundation classes to access CICS services.

Please refer to the reference section for further information on the **lccBuf** class. You might also find the supplied sample – ICC\$BUF – helpful.

Chapter 7. Using CICS Services

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

- “File control”
- “Program control” on page 32
- “Starting transactions asynchronously” on page 34
- “Transient Data” on page 37
- “Temporary storage” on page 39
- “Terminal control” on page 42
- “Time and date services” on page 44

File control

The file control classes – **IccFile**, **IccFileld**, **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 **IccFileld** 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 topic section describes writing records that have not previously been written. Writing records that already exist is not permitted unless they have been previously been put into 'update' mode. See "Updating records" on page 29 for more information.

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

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

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

Writing KSDS records

You must use a key, held in an **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".

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

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

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

Deleting records

Records can never be deleted from an ESDS file.

Deleting normal records

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

Deleting locked records

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

Browsing records

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

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

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

Example of file control

This sample program demonstrates how to use the **IccFile** and **IccFileIterator** classes. The source for this sample can be found in 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.

```

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 303 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 171 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.

```
return;  
};
```

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

See Appendix C, “Output from sample programs,” on page 303 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).

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 "iccmmain.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.

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 trm = [%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 303 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 (**writeltem** 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 39.

Reading data

The **readItem** method is used to read items from the queue. It returns a reference to the **IccBuf** object that contains the information.

Writing data

The **writelnItem** 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 "iccmmain.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 303 for the expected output from this sample program.

Temporary storage

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

You can:

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

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

The methods available in **IccTempStore** class are similar to those in the **IccDataQueue** class. For more information on these see "Transient Data" on page 37.

Reading items

The **readItem** method of **IccTempStore** reads the specified item from the temporary storage queue. It returns a reference to the **IccBuf** object that contains the information.

Writing items

Writing items is also known as "adding" items. This section describes writing items that have not previously been written. Writing items that already exist can be done using the **rewriteItem** method. See "Updating items" for more information.

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

Updating items

Updating an item is also known as "rewriting" an item. The **rewriteItem** method of **IccTempStore** class is used to update the specified item in the temporary storage queue.

Deleting items

You cannot delete individual items in a temporary storage queue. To delete *all* the temporary data associated with an **IccTempStore** object use the **empty** method of **IccTempStore** class.

Example of Temporary Storage

This sample program demonstrates how to use the **IccTempStore** and **IccTempStoreId** classes. This program 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 303 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.

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 Chapter 55, "IccTerminalData class," on page 259 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 303 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 "iccmmain.hpp"
void IccUserControl::run()
{
```

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

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

```
IccClock clock;
```

This creates a clock object.

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

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

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

This fragment demonstrates the use of the **daysSince1900** and **dayOfWeek** methods. **dayOfWeek** returns an enumeration that indicates the day of the week. If

it is Friday, a message is sent to the screen, 'Today IS Friday'; otherwise the message 'Today is NOT Friday' is sent.

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

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

```
term->sendLine( "time() = [%s]",
               clock.time() );
term->sendLine( "time('-') = [%s]",
               clock.time('-') );
term->sendLine( "year() = [%1d]",
               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 303 for the expected output from this sample program.

Chapter 8. Compiling, executing, and debugging

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

Compiling Programs

To compile and link a CICS Foundation Class program you need access to the 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 "iccmmain.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$HEL JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid  
//PROCLIB JCLLIB ORDER=(CICSTS32.CICS.SDFHPROC)  
//ICC$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
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 "iccmmain.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 51
- “Platform differences” on page 54

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, GC34-6827*.

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

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

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

```

void IccUserControl::run()
{
    IccTerminal* term = IccTerminal::instance();
    term->erase();
    *term << "This is program 'icc$excl' ..." << endl;
    try {
        Test test;
        test.tryNumber( 1 );
        test.tryNumber( 7 );
        test.tryNumber( 11 );
        test.tryNumber( 6 );
    }
    catch( const char* exception ) {
        term->setLine( 22 );
        *term << "Exception caught: " << exception << endl << flush;
    }
    term->send( 24,1,"Program 'icc$excl' 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$excl' ...
Number passed = 1
Number passed = 7
Number passed = 11
>>Out of Range - throwing exception
Exception caught: !!Number is out of range!!
Program 'icc$excl' 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 "iccmmain.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 303 for the expected output from this sample program.

invalidMethodCall

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

CICSCondition

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

familyConformanceError

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

internalError

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

CICS conditions

The CICS foundation classes provide a powerful framework for handling conditions that happen when executing an application. Accessing a CICS resource can raise a number of CICS conditions as documented in Foundation Classes—reference.

A condition might represent an error or 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 'callHandleEvent' for a particular CICS condition.

Exception handling (throwException)

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

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

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

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

An exception is thrown if any of the methods inside the try block raise the QIDERR condition for object 'temp'. When an exception is thrown, C++++ unwinds the stack and resumes execution at an appropriate **catch** block – it is not possible to resume within the **try** block. For a fuller example 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 49 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 topic 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 Foundation Classes—reference . Platform variations exist at:

- object level
- method level
- parameter level

Object level

Some objects are not supported on certain platforms. For example **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 journal7(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 66) the previous examples both cause an **IccException** object, of type 'familyConformanceError' to be thrown on all platforms.

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

Method level

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.

```
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

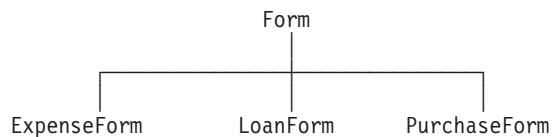
This chapter describes the following:

- “Polymorphic Behavior”
- “Storage management” on page 59
- “Parameter passing conventions” on page 60
- “Scope of data in lccBuf reference returned from 'read' methods” on page 61

Polymorphic Behavior

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

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



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

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

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

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

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

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

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

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

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

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

These virtual methods are **not** supported by any subclasses of **IccResource** except those in the table 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 "iccmmain.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("ICCO");
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 13. Objects on the stack are automatically destroyed when they go out of scope, but objects on the heap are not.

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

On CICS Transaction Server for OS/390, CICS and Language Environment® manage **all** task storage so that it is released at task termination (normal or abnormal).

In a CICS for OS/2 or CICS for AIX environment, as in the earlier Technology Releases for those platforms, storage allocated on the heap is *not* automatically released at task termination. This can lead to "memory leaks" if the application programmer forgets to explicitly delete an object on the heap, or, more seriously, if the task abends.

This problem has been overcome in the CICS Foundation Classes by providing operators **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 287.)

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" ICCMAIN as follows:

```
#define ICC_CLASS_MEMORY_MGMT Icc::cmmNonCICS
#include "icmain.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).

Part 3. Foundation Classes—reference

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

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

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

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

test

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

set

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

catchException

This is the function of last resort, used to intercept **IccException** objects that the application fails to catch. It can be called from the **main** function in the stub program, listed in ICCMAIN header file, and described in Chapter 64, "main function," on page 287. All OO CICS programs should use this stub or a close equivalent.

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

exception

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

conditionText

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

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

condition

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

initializeEnvironment

Initializes the CICS Foundation Classes. The rest of the class library can only be called after this function has been called. It is called from the **main** function in the stub program, listed in ICCMAIN header file, and described in Chapter 64, “main function,” on page 287. All OO CICS programs should use this stub or a close equivalent.

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

mem

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

fam

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

EDF

A boolean that indicates whether EDF tracing is initially on.

isClassMemoryMgmtOn

```
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 Chapter 42, “IccResource class,” on page 183 and “**Execution Diagnostic Facility**” on page 48).

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

setEDF

Sets EDF tracing on or off at the global level.

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

onOff

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

unknownException

```
static void unknownException()
```

This function is called by the **main** function in ICCMAIN header file (see Chapter 64, “main function,” on page 287) and is used to intercept unknown exceptions. (See also **catchException** in this structure).

Enumerations

Note: References in this topicsection 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

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 **IccBase** class map storage allocations to CICS; storage is automatically released at task termination.

FamilySubset

fsDefault

The defaults for the different platforms are all the same:

fsAllowPlatformVariance

fsEnforce

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

fsAllowPlatformVariance

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

GetOpt

This enumeration is used on a number of methods throughout the classes.

It indicates whether the value held internally by the object is to be returned to the caller, or whether it has to be refreshed from CICS first.

object

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

CICS

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

Platforms

Indicates on which operating system the program is being run. Possible values are:

- OS2
- UNIX
- MVS

Chapter 12. IccAbendData class

IccBase
IccResource
IccAbendData

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

Header file: ICCABDEH

IccAbendData constructor (protected)

Constructor

IccAbendData()

Public methods

The *opt* parameter

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

abendCode

Returns the current 4-character abend code.

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

opt

An enumeration, defined in the **Icc** structure, that indicates whether a value should be refreshed from CICS or whether the existing value should be retained. The possible values are described under the **GetOpt** enumeration in the **Icc** structure in topic on page “GetOpt” on page 68.

Conditions

INVREQ

ASRAInterrupt

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

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

Conditions

INVREQ

ASRAKeyType

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

The possible values are:

CICSEXECKEY

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

USEREXECKEY

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

NONCICS

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

NOTAPPLIC

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

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

Conditions

INVREQ

ASRAPSW

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

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

Conditions

INVREQ

ASRARegisters

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

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

Conditions

INVREQ

ASRASpaceType

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

Possible values are:

SUBSPACE

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

BASESPACE

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

NOTAPPLIC

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

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

Conditions

INVREQ

ASRAStorageType

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

Possible values are:

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

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

USER

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

READONLY

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

NOTAPPLIC

One of:

- No ASRA or AEYD abend has been found for this task.
- The storage affected by an abend is not managed by CICS.

- The ASRA abend is not caused by a OC4 abend.
- An ASRB or AICA abend has occurred since the last ASRA or AEYD abend.

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

Conditions

INVREQ

instance

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

static lccAbendData* instance()

isDumpAvailable

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

Returns a boolean, defined in **lcc** 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

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

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

Conditions

INVREQ

programName

Returns the name of the program that caused the abend.

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

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 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)

The copy constructor.

IccAbsTime(const IccAbsTime& *time*)

Public methods

date

Returns the date, as a character string.

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

format

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

dateSeparator

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

Conditions

INVREQ

dayOfMonth

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

unsigned long dayOfMonth()

Conditions

INVREQ

dayOfWeek

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

IccClock::DayOfWeek dayOfWeek()

Conditions

INVREQ

daysSince1900

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

unsigned long daysSince1900()

Conditions

INVREQ

hours

Returns the hours component of the time.

virtual unsigned long hours() const

milliSeconds

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

lccClock::MonthOfYear monthOfYear()

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

Conditions

INVREQ

operator=

Assigns one **lccAbsTime** object to another.

lccAbsTime& operator=(const lccAbsTime& absTime)

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

Returns the time as a text string.

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

timeSeparator

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

Conditions

INVREQ

timeInHours

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

unsigned long timeInHours()

timeInMinutes

unsigned long timeInMinutes()

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

timeInSeconds

unsigned long timeInSeconds()

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

year

unsigned long year()

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

Conditions

INVREQ

Inherited public methods

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

setClassName
setCustomClassNum

Class

IccBase
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)

Creates a new object with no information present.

`IccAlarmRequestId()`

Constructor (2)

Creates an object with information already set.

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

name

The 8-character name of the request.

timerECA

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

Constructor (3)

The copy constructor.

`IccAlarmRequestId(const IccAlarmRequestId& id)`

id A reference to an `IccAlarmRequestId` object.

Public methods

`isExpired`

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

`Icc::Bool isExpired()`

operator= (1)

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

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

operator= (2)

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

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

operator= (3)

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

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

requestName

The 8-character name of the alarm request.

setTimerECA

`void setTimerECA(const void* timerECA)`

timerECA

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

timerECA

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

`const void* timerECA() const`

Inherited public methods

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>
<code>operator new</code>	<code>IccBase</code>

Inherited protected methods

Method

operator=
setClassName
setCustomClassNum

Class

IccResourceId
IccBase
IccBase

Chapter 15. IccBase class

IccBase

IccBase class is the base class from which *all* CICS Foundation Classes are derived. (The methods associated with **IccBase** are described here although, in practice, they can only be called on objects of the derived classes).

Header file: ICCBASEH

IccBase constructor (protected)

Constructor

IccBase(ClassType *type*)

type

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

Public methods

The *opt* parameter

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

classType

Returns an enumeration that indicates what the subclass type is. For example, for an **IccTempStore** object, the class type is 'cTempStore'. The possible values are listed under **ClassType** on page "ClassType" on page 89.

ClassType classType() const

className

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

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

opt

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

customClassNum

unsigned short customClassNum() const

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

operator delete

Destroys an object in an orderly manner.

void operator delete(void* object)

object

A pointer to an object that is to be destroyed.

operator new

Creates a new object of given size. This operator enables the Foundation Classes to use CICS storage allocation (see "initializeEnvironment" on page 66).

void* operator new(size_t size)

size

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

Protected methods

setClassName

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

void setClassName(const char* className)

className

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

setCustomClassNum

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

void setCustomClassNum(unsigned short number)

number

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

Enumerations

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 87.

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 23.

Header file: `ICCBUF.H`

Sample: `ICC$BUF`

`IccBuf` constructors

Constructor (1)

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

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

length

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

type

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

Constructor (2)

Creates an `IccBuf` object that cannot be extended, adopting the given data area as its own. See warning about “Internal/External ownership of buffers” on page 23.

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

length

The length of the supplied data area, in bytes

dataArea

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

Constructor (3)

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

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

text

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

type

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

Constructor (4)

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

lccBuf(const lccBuf& *buffer*)

buffer

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

Public methods

append (1)

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

**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.

append (2)

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

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

format

The null-terminated format string

... The optional parameters.

assign (1)

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

**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.

assign (2)

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

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

format

The format string

... The optional parameters.

cut

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

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

length

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

offset

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

dataArea

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

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

offset

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

dataAreaLength

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 99.

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 99.

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

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

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

length

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

dataArea

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

offset

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

isFMHContained

Icc::Bool isFMHContained() const

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

operator const char*

operator const char*() const

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

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

operator= (1)

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

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

buffer

A reference to an **IccBuf** object.

operator= (2)

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

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

text

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

operator+= (1)

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

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

buffer

A reference to an **IccBuf** object.

operator+= (2)

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

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

text

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

operator==

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

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

buffer

A reference to an **IccBuf** object.

operator!=

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

lcc::Bool operator!=(const lccBuf& *buffer*) const

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

operator« (1)

operator«(const lccBuf& *buffer*)

Appends another buffer.

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

Makes the data area external and fixed. Any existing internal data area is destroyed. See warning about “Internal/External ownership of buffers” on page 23.

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

length

The length of the existing data area.

dataArea

The address of the existing data area.

replace

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

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

length

The length of the source data area, in bytes.

dataArea

The address of the start of the source data area.

offset

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

setDataLength

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

unsigned long setDataLength(unsigned long *length*)

length

The new length of the data area, in bytes

setFMHContained

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

```
void setFMHContained(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.

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 **lccBuf** object has been allocated outside the object. Possible values are:

internal

The data area has been allocated by the **lccBuf** constructor.

external

The data area has been allocated externally.

DataAreaType

Indicates whether the data area of a **lccBuf** 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

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

IccAbsTime& absTime()

cancelAlarm

Cancels a previous **setAlarm** request if the alarm time has not yet been reached, that is, the request has not expired.

void cancelAlarm(const IccRequestId* *reqId* = 0)

reqId

An optional pointer to the **IccRequestId** object that holds information on an alarm request.

Conditions

ISCINVREQ, NOTAUTH, NOTFND, SYSIDERR

date

Returns the date as a string.

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

format

An enumeration, defined in this class, that indicates in which format you want the date to be returned.

dateSeparator

The character that is used to separate different fields in the date. The default is no separation character.

Conditions

INVREQ

dayOfMonth

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

```
unsigned long dayOfMonth()
```

Conditions

INVREQ

dayOfWeek

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

```
DayOfWeek dayOfWeek()
```

Conditions

INVREQ

daysSince1900

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

```
unsigned long daysSince1900()
```

Conditions

INVREQ

milliseconds

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

`long double milliSeconds()`

`monthOfYear`

`MonthOfYear monthOfYear()`

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

Conditions

INVREQ

`setAlarm`

Sets an alarm at the time specified in *time*. It returns a reference to an **lccAlarmRequestId** object that can be used to cancel the alarm—see **cancelAlarm** method. See also the **waitOnAlarm** method on page “waitOnAlarm” on page 230 of class **lccTask**.

```
const lccAlarmRequestId& setAlarm (const lccTime& time,  
                                const lccRequestId* reqId = 0)
```

time

A reference to an **lccTime** object that contains time information. As **lccTime** is an abstract class *time* is, in practise, an object of class **lccAbsTime**, **lccTimeOfDay**, or **lccTimeInterval**.

reqId

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

Conditions

EXPIRED, INVREQ

`time`

Returns the time as a text string.

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

timeSeparator

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

Conditions

INVREQ

update

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

void update()

year

unsigned long year()

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

Conditions

INVREQ

Inherited public methods

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

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

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

IccConsole()

Public methods

The opt parameter

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

instance

Returns a pointer to the single **IccConsole** object that represents the CICS console. If the object does not already exist, it is created by this method.

static IccConsole* instance()

put

Writes the data in *send* to the CICS console. **put** is a synonym for **write**. See “Polymorphic Behavior” on page 57.

virtual void put(const IccBuf& send)

send

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

replyTimeout

unsigned long replyTimeout() const

Returns the length of the reply timeout in milliseconds.

resetRouteCodes

void resetRouteCodes()

Removes all route codes held in the **IccConsole** object.

setAllRouteCodes

void setAllRouteCodes()

Sets all possible route codes in the **IccConsole** object, that is, 1 through 28.

setReplyTimeout (1)

void setReplyTimeout(IccTimeInterval& interval)

interval

A reference to a **IccTimeInterval** object that describes the length of the time interval required.

setReplyTimeout (2)

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

void setReplyTimeout(unsigned long seconds)

seconds

The length of the time interval required, in seconds.

setRouteCodes

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

**void setRouteCodes (unsigned short numRoutes,
...)**

numRoutes

The number of route codes provided in this call—the number of arguments that follow this one.

... One or more arguments, the number of which is given by *numRoutes*. Each argument is a route code, of type **unsigned short**, in the range 1 to 28.

write

Writes the data in *send* to the CICS console.

**void write (const IccBuf& send,
SeverityOpt opt = none)**

send

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

opt

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

Conditions

INVREQ, LENGERR, EXPIRED

writeAndGetReply

Writes the data in *send* to the CICS console and returns a reference to an **IccBuf** object that contains the reply from the CICS operator.

**const IccBuf& writeAndGetReply (const IccBuf& send,
SeverityOpt opt= none)**

send

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

opt

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

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

Returns a reference to an **IccProgramId** object that represents the program that called this program. The returned **IccProgramId** reference contains a null name if the executing program was not called by another program.

const IccProgramId& callingProgramId()

Conditions

INVREQ

cancelAbendHandler

Cancels a previously established exit at this logical program level.

void cancelAbendHandler()

Conditions

NOTAUTH, PGMIDERR

commArea

Returns a reference to an **IccBuf** object that encapsulates the COMMAREA—the communications area of CICS memory that is used for passing data between CICS programs and transactions.

IccBuf& commArea()

Conditions

INVREQ

console

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

IccConsole* console()

initData

const IccBuf& initData()

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

Conditions

INVREQ

instance

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

static IccControl* instance()

isCreated

static Icc::Bool isCreated()

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

programId

const IccProgramId& programId()

Returns a reference to an **IccProgramId** object that refers to this executing program.

Conditions

INVREQ

resetAbendHandler

Reactivates a previously cancelled abend handler for this logical program level. (See **cancelAbendHandler** on page “cancelAbendHandler” on page 113).

void resetAbendHandler()

Conditions

NOTAUTH, PGMIDERR

returnProgramId

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

const IccProgramId& returnProgramId()

run

virtual void run() = 0

This method should be implemented in a subclass of **IccControl** by the application programmer.

session

IccSession* session()

Returns a pointer to the **IccSession** object that represents the principal facility for this program. An exception is thrown if this program does not have a session as its principal facility.

setAbendHandler (1)

void setAbendHandler(const IccProgramId& *programId*)

programId

A reference to the **IccProgramId** object that indicates which program is affected.

setAbendHandler (2)

These methods set the abend handler to the named program for this logical program level.

void setAbendHandler(const char* *programName*)

programName

The name of the program affected.

Conditions

NOTAUTH, PGMIDERR

startRequestQ

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

IccStartRequestQ* startRequestQ()

system

IccSystem* system()

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

task

IccTask* task()

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

terminal

IccTerminal* terminal()

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

This method has a condition, that the transaction must have a terminal as its principal facility. That is, there must be a physical terminal involved.

Inherited public methods

Method

actionOnCondition
actionOnConditionAsChar
actionsOnConditionsText
classType
className
condition
conditionText
customClassNum
handleEvent
id
isEDFOn
name
operator delete

Class

IccResource
IccResource
IccResource
IccBase
IccBase
IccResource
IccResource
IccBase
IccResource
IccResource
IccResource
IccResource
IccResource
IccBase

Method

operator new
setActionOnAnyCondition
setActionOnCondition
setActionsOnConditions
setEDF

Class

IccBase
IccResource
IccResource
IccResource
IccResource

Inherited protected methods**Method**

setClassName
setCustomClassNum

Class

IccBase
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)

The copy constructor.

IccConvId(const IccConvId& convId)

convId
A reference to an IccConvId object.

Public methods

operator= (1)

IccConvId& operator=(const char* convName)

operator= (2)

Assigns new value.

IccConvId& operator=(const IccConvId id)

Inherited public methods

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

Method
operator new

Class
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

A synonym for **empty**. See “Polymorphic Behavior” on page 57.

virtual void clear()

empty

void empty()

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

Conditions

ISCVREQ, NOTAUTH, QIDERR, SYSIDERR, DISABLED, INVREQ

get

A synonym for **readItem**. See “Polymorphic Behavior” on page 57.

virtual const lccBuf& get()

put

A synonym for **writeltem**. See “Polymorphic Behavior” on page 57.

virtual void put(const lccBuf& buffer)

buffer

A reference to an **lccBuf** object that contains data to be put into the queue.

readItem

const lccBuf& readItem()

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

Conditions

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

writeltem (1)

void writeltem(const lccBuf& item)

item

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

writeltem (2)

Writes an item of data to the queue.

void writeltem(const char* text)

text

Text that is to be written to the queue.

Conditions

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

Inherited public methods

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

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)

Assigns new value.

IccDataQueueId& operator=(const IccDataQueueId& id)

id A reference to an IccDataQueueId object.

Inherited public methods

Method	Class
classType	IccBase
className	IccBase
customClassNum	IccBase

Method
name
nameLength
operator delete
operator new

Class
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

Returns the name of the class responsible for this event.

const char* **className()** **const**

classType

IccBase::ClassType **classType()** **const**

Returns an enumeration, described under **classType** on page “classType” on page 87 in **IccBase** class, that indicates the type of class that is responsible for this event.

condition

Returns an enumerated type that indicates the condition returned from this CICS event. The possible values are described under the **Codes** type in the **IccCondition** structure.

IccCondition::Codes **condition(IccResource::ConditionType** *type* =
IccResource::majorCode) **const**

type

An enumeration that indicates whether a major code or minor code is being requested. Possible values are 'majorCode' or 'minorCode'. 'majorCode' is the default value.

conditionText

const char* conditionText() const

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

methodName

const char* methodName() const

Returns the name of the method responsible for this event.

summary

const char* summary()

Returns a summary of the CICS event in the form:

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

Inherited public methods

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

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

`const char* className() const`

classType

`IccBase::ClassType classType() const`

Returns an enumeration, described under **ClassType** in **IccBase** class, that indicates the type of class which threw this exception.

message

`IccMessage* message() const`

Returns a pointer to an **IccMessage** object that contains information on any message associated with this exception.

methodName

`const char* methodName() const`

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

number

`unsigned short number() const`

Returns the unique exception number.

This is a useful diagnostic for IBM service. The number uniquely identifies from where in the source code the exception was thrown.

summary

`const char* summary()`

Returns a string containing a summary of the exception. This combines the **className**, **methodName**, **number**, **Type**, and **IccMessage::text** methods into the following form:

CICS exception summary: 094 IccTempStore::readNextItem type=CICSCondition

type

Type type() const

Returns an enumeration, defined in this class, that indicates the type of exception.

typeText

const char* typeText() const

Returns a string representation of the exception type, for example, "objectCreationError", "invalidArgument".

Inherited public methods

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

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. An attempt to create an **IccFileId** object with a 9-character filename also generates an exception of this type.

invalidMethodCall

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

CICSCondition

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

platformError

An operation is invalid because of limitations of this particular platform. For example, an attempt to create an **IccJournal** object would fail under CICS for OS/2 because there are no CICS journal services on this server.

A platformError exception can occur at 3 levels:

1. An object is not supported on this platform.
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 54 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)

To access files using an **IccFile** object, it must have an **IccRecordIndex** object associated with it. If this association is not made when the object is created, use the **registerRecordIndex** method.

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

fileName

The 8-character name of the file

index

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

Public methods

The opt parameter

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

access

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

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

opt

An enumeration, defined in **Icc** structure, that indicates whether you can use a value previously retrieved from CICS (object), or whether the object should retrieve a fresh value from CICS.

accessMethod

Returns an enumeration, defined in **IccValue**, that represents the access method for this file.

Possible values are:

- VSAM
- BDAM
- SFS

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

opt

See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

beginInsert(VSAM only)

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

void beginInsert()

deleteLockedRecord

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

void deleteLockedRecord(unsigned long updateToken = 0)

updateToken

A token that indicates which previously read record is to be deleted. This is the token that is returned from **readRecord** method when in update mode.

Conditions

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

deleteRecord

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

unsigned short deleteRecord()

Conditions

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

enableStatus

Returns an enumeration, defined in **IccValue**, that indicates whether the file is enabled to be used by programs.

Possible values are:

- DISABLED
- DISABLING
- ENABLED
- UNENABLED
- UNENABLING

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

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

endInsert(VSAM only)

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

void endInsert()

isAddable

Indicates whether more records can be added to the file.

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

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isBrowsable

Indicates whether the file can be browsed.

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

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isDeletable

Indicates whether the records in the file can be deleted.

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

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isEmptyOnOpen

Returns a Boolean that indicates whether the EMPTYREQ option is specified. EMPTYREQ causes the object associated with this file to be set to empty when opened, if it is a VSAM data set defined as reusable.

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

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isReadable

Indicates whether the file records can be read.

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

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

isRecoverable

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

opt
See **access** method.

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

isUpdatable

Indicates whether the file can be updated.

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

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

keyLength

Returns the length of the search key.

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

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

keyPosition

Returns the position of the key field in each record relative to the beginning of the record. If there is no key, zero is returned.

long keyPosition(lcc::GetOpt opt = lcc::object)

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

openStatus

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

lccValue::CVDA openStatus(lcc::GetOpt *opt* = lcc::object)

opt

See **access** method.

CLOSED

The file is closed.

CLOSING

The file is in the process of being closed. Closing a file may require dynamic deallocation of data sets and deletion of shared resources, so the process may last a significant length of time.

CLOSEREQUEST

The file is open and one or more application tasks are using it. A request has been received to close it.

OPEN

The file is open.

OPENING

The file is in the process of being opened.

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

readRecord

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

**const lccBuf& 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

Conditions

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

recordFormat

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

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

opt
See **access** method.

FIXED

The records are of fixed length.

UNDEFINED (BDAM data sets only)

The format of records on the file is undefined.

VARIABLE

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

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

recordIndex

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

IccRecordIndex* recordIndex() const

recordLength

Returns the length of the current record.

unsigned long recordLength(Icc::GetOpt *opt* = Icc::object)

opt
See **access** method.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

registerRecordIndex

void registerRecordIndex(IccRecordIndex* *index*)

index
A pointer to an **IccKey**, **IccRBA**, or **IccRRN** object that will be used by methods such as **readRecord**, **writeRecord**, etc..

rewriteRecord

Updates a record with the contents of *buffer*.

void rewriteRecord (const IccBuf& *buffer*,
unsigned long *updateToken* = 0)

buffer
A reference to the **IccBuf** object that holds the new record data to be written to the file.

updateToken

The token that identifies which previously read record is to be rewritten. See **readRecord**.

Conditions

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

setAccess

Sets the permitted access to the file.

For example:

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

void setAccess(unsigned long access)

access

A positive integer value created by ORing (or adding) one or more of the values of the Access enumeration, defined in this class.

Conditions

FILENOTFOUND, INVREQ, IOERR, NOTAUTH

setEmptyOnOpen

void setEmptyOnOpen(Icc::Bool trueFalse)

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

Conditions

FILENOTFOUND, INVREQ, IOERR, NOTAUTH

setStatus

Sets the status of the file.

void setStatus(Status status)

status

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

Conditions

FILENOTFOUND, INVREQ, IOERR, NOTAUTH

type

Returns a CVDA that identifies the type of data set that corresponds to this file. Possible values are:

lccValue::CVDA type(**lcc::GetOpt** *opt* = **lcc::object**)

opt

See **access** method.

ESDS

The data set is an entry-sequenced data set.

KEYED

The data set is addressed by physical keys.

KSDS

The data set is a key-sequenced data-set.

NOTKEYED

The data set is not addressed by physical keys.

RRDS

The data set is a relative record data set.

VRRDS

The data set is a variable relative record data set.

Conditions: END, FILENOTFOUND, ILLOGIC, NOTAUTH

unlockRecord

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

void unlockRecord(unsigned long updateToken = 0)

updateToken

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

Conditions

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

writeRecord

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

void writeRecord(const lccBuf& buffer)

buffer

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

Conditions

DISABLED, DUPREC, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISINVREQ, LENGERR, NOSPACE, NOTAUTH, NOTOPEN, SYSIDERR, LOADING, SUPPRESSED

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

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:

- it is rewritten using the **rewriteRecord** method *or*
- it is deleted using the **deleteLockedRecord** method *or*
- it is unlocked using the **unlockRecord** method *or*
- the task commits or rolls back its resource updates *or*
- the task is abended.

SearchCriterion

equalToKey

The search only finds an exact match.

gteqToKey

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

Status

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

closed

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

enabled

File is enabled for access by CICS tasks.

disabled

File is disabled from access by CICS tasks.

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)

Assigns new value.

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

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

Inherited public methods

Method	Class
classType	IccBase
className	IccBase

Method

customClassNum
name
nameLength
operator delete
operator new

Class

IccBase
IccResourceId
IccResourceId
IccBase
IccBase

Inherited protected methods**Method**

operator=
setClassName
setCustomClassNum

Class

IccResourceId
IccBase
IccBase

Chapter 28. IccFileIteerator class

IccBase
IccResource
IccFileIteerator

This class is used to create **IccFileIteerator** 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

IccFileIteerator constructor

Constructor

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

IccFileIteerator (**IccFile*** *file*,
IccRecordIndex* *index*,
IccFile::SearchCriterion *search* = IccFile::gteqToKey)

file

A pointer to the **IccFile** object that is to be browsed

index

A pointer to the **IccRecordIndex** object that is being used to select a record in the file

search

An enumeration, defined in **IccFile**, that indicates the criterion being used to find a search match. The default is gteqToKey.

Conditions

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

Public methods

readNextRecord

Read the record that follows the current record.

const IccBuf& readNextRecord (**IccFile::ReadMode** *mode* = IccFile::normal,
unsigned long* *updateToken* = 0)

mode

An enumeration, defined in **IccFile** class, that indicates the type of read request

updateToken

A returned token that is used to identify this unique update request on a subsequent **rewriteRecord**, **deleteLockedRecord**, or **unlockRecord** method on the file object.

Conditions

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

readPreviousRecord

Read the record that precedes the current record.

```
const IccBuf& readPreviousRecord (IccFile::ReadMode mode = IccFile::normal,  
                                unsigned long* updateToken = 0)
```

mode

An enumeration, defined in **IccFile** class, that indicates the type of read request.

updateToken

See **readNextRecord**.

Conditions

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

reset

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

```
void reset (IccRecordIndex* index,  
           IccFile::SearchCriterion search = IccFile::gteqToKey)
```

index

A pointer to the **IccRecordIndex** object that is being used to select a record in the file.

search

An enumeration, defined in **IccFile**, that indicates the criterion being used to find a search match. The default is **gteqToKey**.

Conditions

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

Inherited public methods

Method

actionOnCondition
actionOnConditionAsChar

Class

IccResource
IccResource

Method	Class
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

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)

The copy constructor.

IccGroupId(const IccGroupId& *id*)

id A reference to an IccGroupId object.

Public methods

operator= (1)

IccGroupId& operator=(const char* *groupName*)

groupName
The 8-character name of the group.

operator= (2)

Assigns new value.

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

id A reference to an IccGroupId object.

Inherited public methods

Method	Class
classType	IccBase
className	IccBase

Method

customClassNum
name
nameLength
operator delete
operator new

Class

IccBase
IccResourceId
IccResourceId
IccBase
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`

Clears the current prefix as set by `registerPrefix` or `setPrefix`. If the current prefix was set using `registerPrefix`, then the `IccJournal` class only removes its own reference to the prefix. The buffer itself is left unchanged. If the current prefix was set by `setPrefix`, then the `IccJournal`'s copy of the buffer is deleted.

void clearPrefix()

journalTypeId

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

const IccJournalTypeId& journalTypeId() const

put

A synonym for **writeRecord**—puts data into the journal. See “Polymorphic Behavior” on page 57 for information on polymorphism.

virtual void put(const IccBuf& buffer)

buffer

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

registerPrefix

void registerPrefix(const IccBuf* prefix)

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

setJournalTypeId (1)

void setJournalTypeId(const IccJournalTypeId& id)

setJournalTypeId (2)

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 IccBuf& prefix)

setPrefix (2)

void setPrefix(const char* prefix)

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

wait

Waits until a previous journal write has completed.

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

requestNum

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

option

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

writeRecord (1)

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

record

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

option

See above.

writeRecord (2)

Writes the data in the record to the journal. The returned number represents the particular write request and can be passed to the **wait** method in this class.

```
unsigned long writeRecord (const char* record,  
                          unsigned long option = 0)
```

record

The name of the record

option

See above.

Conditions

IOERR, JIDERR, LENGERR, NOJBUFSP, NOTAUTH, NOTOPEN

Inherited public methods

Method

actionOnCondition
actionOnConditionAsChar
actionsOnConditionsText
classType
className

Class

IccResource
IccResource
IccResource
IccBase
IccBase

Method	Class
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

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)

The copy constructor.

`IccJournalId(const IccJournalId& id)`

id A reference to an `IccJournalId` object.

Public methods

`number`

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

`unsigned short number() const`

`operator= (1)`

`IccJournalId& operator=(unsigned short journalNum)`

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

`operator= (2)`

Assigns new value.

`IccJournalId& operator=(const IccJournalId& id)`

id A reference to an `IccJournalId` object.

Inherited public methods

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)

Sets the 2-byte field that is included in the journal record.

`void operator=(const char* journalTypeName)`

journalTypeName
A 2-byte identifier used in journal records.

Inherited public methods

Method	Class
<code>classType</code>	<code>IccBase</code>
<code>className</code>	<code>IccBase</code>

Method
customClassNum
name
nameLength
operator delete
operator new

Class
IccBase
IccResourceId
IccResourceId
IccBase
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

Copies the search key into the IccKey object.

```
void assign (unsigned short length,  
            const void* dataArea)
```

length

The length of the data area

dataArea

A pointer to the start of the data area that holds the search key.

completeLength

Returns the length of the key when it is complete.

```
unsigned short completeLength() const
```

kind

Kind kind() const

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

operator= (1)

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

Changes the type of key from generic to complete or vice versa.

`void setKind(Kind kind)`

kind

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

value

`const char* value()`

Returns the start of the data area containing the search key.

Inherited public methods

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.

Chapter 34. `lccLockId` class

`lccBase`
`lccResourceId`
`lccLockId`

`lccLockId` class is used to identify a lock request.

Header file: `ICCRIDEH`

`lccLockId` constructors

Constructor (1)

`lccLockId(const char* name)`

name
The 8-character name of the lock request.

Constructor (2)

The copy constructor.

`lccLockId(const lccLockId& id)`

id A reference to an `lccLockId` object.

Public methods

`operator=` (1)

`lccLockId& operator=(const char* name)`

name
The 8-character name of the lock request.

`operator=` (2)

Assigns new value.

`lccLockId& operator=(const lccLockId& id)`

id A reference to an `lccLockId` object.

Inherited public methods

Method	Class
<code>classType</code>	<code>lccBase</code>
<code>className</code>	<code>lccBase</code>

Method
customClassNum
name
nameLength
operator delete
operator new

Class
IccBase
IccResourceId
IccResourceId
IccBase
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

Returns the name of the class with which the message is associated, if any. If there is no name to return, a null pointer is returned.

const char* className() const

methodName

const char* methodName() const

Returns the name of the method with which the message is associated, if any. If there is no name to return, a null pointer is returned.

number

unsigned short number() const

Returns the number of the message.

summary

const char* summary()

Returns the text of the message.

text

const char* text() const

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

Inherited public methods

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)

The copy constructor.

`IccPartnerId(const IccPartnerId& id)`

id A reference to an `IccPartnerId` object.

Public methods

`operator=` (1)

`IccPartnerId& operator=(const char* partnerName)`

partnerName
The 8-character name of an APPC partner.

`operator=` (2)

Assigns new value.

`IccPartnerId& operator=(const IccPartnerId& id)`

id A reference to an `IccPartnerId` object.

Inherited public methods

Method	Class
<code>classType</code>	<code>IccBase</code>
<code>className</code>	<code>IccBase</code>

Method
customClassNum
name
nameLength
operator delete
operator new

Class
IccBase
IccResourceId
IccResourceId
IccBase
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

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

const void* address() const

clearInputMessage

Clears the current input message which was set by **setInputMessage** or **registerInputMessage**. If the current input message was set using **registerInputMessage** then only the pointer is deleted: the buffer is left unchanged. If the current input message was set using **setInputMessage** then **clearInputMessage** releases the memory used by that buffer.

`void clearInputMessage()`

entryPoint

`const void* entryPoint() const`

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

length

`unsigned long length() const`

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

link

`void link (const 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

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

void registerInputMessage(const IccBuf& msg)

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

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

void unload()

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

Method	Class
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)

Assigns new value.

`IccProgramId& operator=(const IccProgramId& id)`

id A reference to an `IccProgramId` object.

Inherited public methods

Method
classType

Class
`IccBase`

Method

className
customClassNum
name
nameLength
operator delete
operator new

Class

IccBase
IccBase
IccResourceId
IccResourceId
IccBase
IccBase

Inherited protected methods**Method**

operator=
setClassName
setCustomClassNum

Class

IccResourceId
IccBase
IccBase

Chapter 39. IccRBA class

IccBase
IccRecordIndex
IccRBA

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

Header file: ICCRECEH

IccRBA constructor

Constructor

IccRBA(unsigned long *initRBA* = 0)

initRBA

An initial value for the relative byte address.

Public methods

operator= (1)

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

operator= (2)

Assigns a new value for the relative byte address.

IccRBA& operator=(unsigned long *num*)

num

A valid relative byte address.

operator== (1)

Icc::Bool operator== (const IccRBA& *rba*) const

operator== (2)

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

Tests equality

operator!= (1)

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

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

Returns the length of the record identifier.

unsigned short length() const

type

Type type() const

Returns an enumeration, defined in this class, that indicates whether the index type is key, RBA, or RRN.

Inherited public methods

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
- RRR

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)

An empty `IccRequestId` object.

`IccRequestId()`

Constructor (2)

`IccRequestId(const char* requestName)`

requestName
The 8-character name of the request.

Constructor (3)

The copy constructor.

`IccRequestId(const IccRequestId& id)`

id A reference to an `IccRequestId`.

Public methods

`operator=` (1)

`IccRequestId& operator=(const IccRequestId& id)`

id A reference to an `IccRequestId` object whose properties are copied into this object.

`operator=` (2)

Assigns new value.

`IccRequestId& operator=(const char* requestName)`

requestName

An 8-character string which is copied into this object.

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

Returns an enumeration that indicates what action the class will take in response to the specified condition being raised by CICS. The possible values are described in this class.

ActionOnCondition actionOnCondition(IccCondition::Codes condition)

condition

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

actionOnConditionAsChar

char actionOnConditionAsChar(IccCondition::Codes condition)

This method is the same as **actionOnCondition** but returns a character, rather than an enumeration, as follows:

0 (zero)

No action is taken for this CICS condition.

- H** The virtual method **handleEvent** is called for this CICS condition.
- X** An exception is generated for this CICS condition.
- A** This program is abended for this CICS condition.

actionsOnConditionsText

Returns a string of characters, one character for each possible condition. Each character indicates the actions to be performed for that corresponding condition. .

The characters used in the string are described in “actionOnConditionAsChar” on page 183. For example, the string: 0X00H0A . . . shows the actions for the first seven conditions are as follows:

- condition 0 (NORMAL)**
action=0 (noAction)
- condition 1 (ERROR)**
action=X (throwException)
- condition 2 (RDATT)**
action=0 (noAction)
- condition 3 (WRBRK)**
action=0 (noAction)
- condition 4 (ICCEOF)**
action=H (callHandleEvent)
- condition 5 (EODS)**
action=0 (noAction)
- condition 6 (EOC)**
action=A (abendTask)

const char* actionsOnConditionsText()

clear

Clears the contents of the object. This method is virtual and is implemented, wherever appropriate, in the derived classes. See “Polymorphic Behavior” on page 57 for a description of polymorphism. The default implementation in this class throws an exception to indicate that it has not been overridden in a subclass.

virtual void clear()

condition

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

unsigned long condition(**ConditionType** *type* = majorCode) **const**

type

An enumeration, defined in this class, that indicates the type of condition requested. Possible values are majorCode (the default) and minorCode.

conditionText

const char* conditionText() const

Returns the symbolic name of the last CICS condition for this object.

get

virtual const IccBuf& get()

Gets data from the **IccResource** object and returns it as an **IccBuf** reference. This method is virtual and is implemented, wherever appropriate, in the derived classes. See “Polymorphic Behavior” on page 57 for a description of polymorphism. The default implementation in this class throws an exception to indicate that it has not been overridden in a subclass.

handleEvent

This virtual function may be re-implemented in a subclass (by the application programmer) to handle CICS events (see **IccEvent** class on page Chapter 24, “IccEvent class,” on page 127).

virtual HandleEventReturnOpt handleEvent(IccEvent& event)

event

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

id

const IccResourceId* id() const

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

isEDFOn

Icc::Bool isEDFOn() const

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

isRouteOptionOn

Icc::Bool isRouteOptionOn() const

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

name

const char* name() const

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

put

Puts information from the buffer into the **IccResource** object. This method is virtual and is implemented, wherever appropriate, in the derived classes. See “Polymorphic Behavior” on page 57 for more information on polymorphism. The default implementation in this class throws an exception to indicate that it has not been overridden in a subclass.

virtual void put(const IccBuf& buffer)

buffer

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

routeOption

const IccSysId& routeOption() const

Returns a reference to an **IccSysId** object that represents the system to which all CICS requests are routed—explicit function shipping.

setActionOnAnyCondition

Specifies the default action to be taken by the CICS foundation classes when a CICS condition occurs.

void setActionOnAnyCondition(ActionOnCondition action)

action

The name of the action as an enumeration. The possible values are listed under the description of this class.

setActionOnCondition

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

**void setActionOnCondition (ActionOnCondition action,
IccCondition::Codes condition)**

action

The name of the action as an enumeration. The possible values are listed under the description of this class.

condition

See **IccCondition** structure.

setActionsOnConditions

void setActionsOnConditions(const char* actions = 0)

actions

A string that indicates what action is to be taken for each condition. The default is not to indicate any actions, in which case each condition is given a default **ActionOnCondition** of **noAction**. The string should have the same format as the one returned by the **actionsOnConditionsText** method.

setEDF

Switches EDF on or off for this resource object. See “Execution Diagnostic Facility” on page 48. These methods force the object to route CICS requests to the named remote system. This is called explicit function shipping.

void setEDF(Icc::Bool onOff)

onOff

A boolean value that selects whether EDF trace is switched on or off.

setRouteOption (1)

void setRouteOption(const IccSysId& sysId)

The parameters are:

sysId

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

setRouteOption (2)

This option is only valid for certain classes: Attempting to use this method on other subclasses of **IccResource** causes an exception to be thrown.

Valid classes are:

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

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

```
obj.setRouteOption()
```

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

sysName

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

Inherited public methods

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 program's responsibility to test CICS conditions using the **condition** method, after executing a method that calls CICS services.

callHandleEvent

Call the virtual **handleEvent** method.

throwException

An **IccException** object is created and thrown. This is typically used for more serious conditions or errors.

abendTask

Abend the CICS task.

HandleEventReturnOpt

Possible values are:

rContinue

The CICS event proceeded satisfactorily and normal processing is to resume.

rThrowException

The application program could not handle the CICS event and an exception is to be thrown.

rAbendTask

The application program could not handle the CICS event and the CICS task is to be abended.

ConditionType

Possible values are:

majorCode

The returned value is the CICS RESP value. This is one of the values in `IccCondition::codes`.

minorCode

The returned value is the CICS RESP2 value.

Chapter 43. IccResourceId class

IccBase
IccResourceId

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

Header file: ICCRIDEH

IccResourceId constructors (protected)

Constructor (1)

IccResourceId (**IccBase::ClassType** *typ*,
const **IccResourceId**& *id*)

type

An enumeration, defined in **IccBase** class, that indicates the type of class.

id A reference to an **IccResourceId** object that is used to create this object.

Constructor (2)

IccResourceId (**IccBase::ClassType** *type*,
const char* *resName*)

type

An enumeration, defined in **IccBase** class, that indicates the type of class.

resName

The name of a resource that is used to create this object.

Public methods

name

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

const char* name() const

nameLength

unsigned short nameLength() const

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

Protected methods

operator=

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

IccResourceId& operator=(const IccResourceId& id)

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

Inherited public methods

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& *rrn*)

operator= (2)

Assigns a new value for the relative record number.

IccRRN& operator=(unsigned long *num*)

num

A relative record number—an integer greater than 0.

operator== (1)

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

operator== (2)

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

Tests equality

operator!= (1)

`lcc::Bool operator!= (const lccRRN& rrn) const`

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	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 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

Returns an enumeration, defined in this class, that indicates whether the lock lasts for the length of the current unit-of-work ('UOW') or until the task terminates('task').

LifeTime **lifeTime()** **const**

lock

`void lock()`

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

Conditions

ENQBUSY, LENGERR, INVREQ

tryLock

Attempts to get a lock. This method does not block if another task already owns the lock. It returns a boolean that indicates whether it succeeded.

`lcc::Bool tryLock()`

Conditions

ENQBUSY, LENGERR, INVREQ

type

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

`LockType type() const`

unlock

`void unlock()`

Release a lock.

Conditions

LENGERR, INVREQ

Inherited public methods

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>
<code>id</code>	<code>IccResource</code>
<code>isEDFOn</code>	<code>IccResource</code>

Method	Class
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

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.

lccSession()

Public methods

allocate

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

void allocate(AllocateOpt option = queue)

option

An enumeration, defined in this class, that indicates what action CICS is to take if a communication channel is unavailable when this method is called.

Conditions

INVREQ, SYSIDERR, CBIDERR, NETNAMEIDERR, PARTNERIDERR, SYSBUSY

connectProcess (1)

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

**void connectProcess (SyncLevel level,
const lccBuf* 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 **lccBuf** object that contains the PIP data to be sent to the remote system

connectProcess (2)

**void connectProcess (SyncLevel level,
const lccTransId& transId,
const lccBuf* 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 **lccTransId** object that holds the name of the transaction to be started on the remote system

PIP

An optional pointer to an **lccBuf** object that contains the PIP data to be sent to the remote system

connectProcess (3)

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

```
void connectProcess (SyncLevel level,  
                    const lccTPNameId& TPName,  
                    const lccBuf* 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 **lccTPNameId** object that contains the 1–64 character TP name.

PIP

An optional pointer to an **lccBuf** object that contains the PIP data to be sent to the remote system

Conditions

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

converse

converse sends the contents of *send* and returns a reference to an **lccBuf** object that holds the reply from the remote APPC partner.

```
const lccBuf& converse(const lccBuf& send)
```

send

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

Conditions

EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

convId

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

```
const lccConvId& convId()
```

errorCode

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

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

extractProcess

void extractProcess()

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

Conditions

INVREQ, NOTALLOC, LENGERR

flush

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

void flush()

Conditions

INVREQ, NOTALLOC

free

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

void free()

Conditions

INVREQ, NOTALLOC

get

A synonym for **receive**. See “Polymorphic Behavior” on page 57 for information on polymorphism.

virtual const lccBuf& get()

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

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

void issueConfirmation()

Conditions

INVREQ, NOTALLOC, TERMERR, SIGNAL

issueError

Signals an error to the partner process.

void issueError()

Conditions

INVREQ, NOTALLOC, TERMERR, SIGNAL

issuePrepare

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

void issuePrepare()

Conditions

INVREQ, NOTALLOC, TERMERR

issueSignal

Signals that a mode change is needed.

void issueSignal()

Conditions

INVREQ, NOTALLOC, TERMERR

PIPList

Returns a reference to an **lccBuf** 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.

lccBuf& PIPList()

process

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

A synonym for **send**. See “Polymorphic Behavior” on page 57 for information on polymorphism.

virtual void put(const lccBuf& data)

data

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

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)

Sends data to the remote partner.

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

option

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

Conditions

INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

sendInvite (1)

```
void sendInvite (const 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 **sendInvite** method. The default is normal.

sendInvite (2)

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

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

option

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

Conditions

INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

sendLast (1)

```
void sendLast (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 **sendLast** method. The default is normal.

sendLast (2)

Sends data to the remote partner and indicates that this is the final transmission. The **free** method must be invoked next, unless the sync level is 2, when you must commit resource updates before the **free**. (See **commitUOW** on page “commitUOW” on page 224 in **lccTaskClass**).

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

option

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

Conditions

INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

state

Returns a CVDA, defined in **lccValue** 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

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

IccValue::CVDA state(StateOpt option = lastCommand)

option

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

Conditions

INVREQ, NOTALLOC

stateText

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

const char* stateText(StateOpt option = lastCommand)

option

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

syncLevel

SyncLevel syncLevel() const

Returns an enumeration, defined in this class, that indicates the synchronization level that is being used in this session. A call to this method should be preceded by a call to **extractProcess** on back end DTP processes.

Inherited public methods

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

AllocateOpt

queue

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

noQueue

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

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

SendOpt

normal

The default.

confirmation

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

wait

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

StateOpt

Used to indicate how the state of a conversation is to be reported.

lastCommand

Return the state at the time of the completion of the last operation on the session.

extractState

Return the explicitly extracted current state.

SyncLevel

level0

Sync level 0

level1

Sync level 1

level2

Sync level 2

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 “start” on page 213).

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

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

```
void cancel (const IccRequestId& reqId,  
            const IccTransId* transId = 0)
```

reqId

A reference to an **IccRequestId** object that represents the request to be cancelled

transId

An optional pointer to an **IccTransId** object that represents the transaction that is to be cancelled.

Conditions

ISCVNREQ, NOTAUTH, NOTFND, SYSIDERR

clearData

```
void clearData()
```

clearData clears the current data that is to be passed to the started transaction. The data was set using **setData** or **registerData**.

If the data was set using **registerData**, only the pointer to the data is removed, the data in the buffer is left unchanged.

If the data was set using **setData**, then **clearData** releases the memory used by the buffer.

data

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

```
const IccBuf& data() const
```

instance

```
static IccStartRequestQ* instance()
```

Returns a pointer to the single **IccStartRequestQ** object. If the object does not exist it is created. See also **startRequestQ** method on page “startRequestQ” on page 116 of **IccControl**.

queueName

```
const char* queueName() const
```

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

registerData

Registers an **IccBuf** object to be interrogated for start data on each subsequent **start** method invocation. This just stores the address of the **IccBuf** object within the **IccStartRequestQ** so that the **IccBuf** object can be found when using the **start** method. This differs from the **setData** method, which takes a copy of the data held in the **IccBuf** object during the time that it is invoked.

```
void registerData(const IccBuf* buffer)
```

buffer

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

reset

```
void reset()
```

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

retrieveData

Used by a task that was started, via an async start request, to gain access to the information passed by the start requester. The information is returned by the **data**, **queueName**, **returnTermId**, and **returnTransId** methods.

```
void retrieveData(RetrieveOpt option = noWait)
```

option

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

Conditions

ENDDATA, ENVDEFERR, IOERR, LENGERR, NOTFND, INVREQ

Note: The ENVDEFERR condition will be raised if all the possible options (**setData**, **setQueueName**, **setReturnTermId**, and **setReturnTransId**) are not used before issuing the **start** method. This condition is therefore not necessarily an error condition and your program should handle it accordingly.

returnTermId

Returns a reference to an **IccTermId** object that identifies which terminal is involved in the session. A call to this method should be preceded by a call to **retrieveData** method.

```
const IccTermId& returnTermId() const
```

returnTransId

```
const IccTransId& returnTransId() const
```

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

setData

```
void setData(const IccBuf& buf)
```

Copies the data in *buf* into the **IccStartRequestQ**, which passes it to the started transaction when the **start** method is called. See also **registerData** on page “registerData” on page 210 for an alternative way to pass data to started transactions.

setQueueName

Requests that this queue name be passed to the started transaction when the **start** method is called.

```
void setQueueName(const char* queueName)
```

queueName
An 8-character queue name.

setReturnTermId (1)

void setReturnTermId(const lccTermId& termId)

termId
A reference to an **lccTermId** object that identifies which terminal is involved in the session.

setReturnTermId (2)

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

void setReturnTermId(const char* termName)

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

setReturnTransId (1)

void setReturnTransId(const lccTransId& transId)

transId
A reference to an **lccTransId** object.

setReturnTransId (2)

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

void setReturnTransId(const char* transName)

transName
The 4-character name of the return transaction.

setStartOpts

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

**void setStartOpts (ProtectOpt popt = none,
 CheckOpt copt = check)**

popt
An enumeration, defined in this class, that indicates whether start requests are to be protected

copt

An enumeration, defined in this class, that indicates whether start requests are to be checked.

start

Asynchronously starts the named CICS transaction. The returned reference to an **IccRequestId** object identifies the **start** request and can be used subsequently to **cancel** the **start** request.

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

or

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

or

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

transId

A reference to an **IccTransId** object that represents the transaction to be started

termId

A reference to an **IccTermId** object that identifies which terminal is involved in the session.

userId

A reference to an **IccUserId** object that represents the user ID.

time

An (optional) pointer to an **IccTime** object that specifies when the task is to be started. The default is for the task to be started immediately.

reqId

An (optional) pointer to an **IccRequestId** object that is used to identify this start request so that the **cancel** can cancel the request.

Conditions

INVREQ, IOERR, ISCVREQ, LENGERR, NOTAUTH, SYSIDERR, TERMIDERR, TRANSIDERR, USERIDERR

Inherited public methods

Method

actionOnCondition
actionOnConditionAsChar

Class

IccResource
IccResource

Method	Class
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

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)

The copy constructor.

`IccSysId(const IccSysId& id)`

id A reference to an `IccSysId` object.

Public methods

`operator=` (1)

`IccSysId& operator=(const IccSysId& id)`

id A reference to an existing `IccSysId` object.

`operator=` (2)

Sets the name of the CICS system held in the object.

`IccSysId& operator=(const char* name)`

name
The 4-character name of the CICS system.

Inherited public methods

Method	Class
<code>classType</code>	<code>IccBase</code>
<code>className</code>	<code>IccBase</code>

Method
customClassNum
name
nameLength
operator delete
operator new

Class
IccBase
IccResourceId
IccResourceId
IccBase
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

Returns the 8-character name of the CICS region.

const char* applName()

Conditions

INVREQ

beginBrowse (1)

void beginBrowse (**ResourceType** resource,
const **IccResourceId*** resId = 0)

resource

An enumeration, defined in this class, that indicates the type of resource to be browsed within the CICS system.

resId

An optional pointer to an **IccResourceId** object that indicates the starting point for browsing through the resources.

beginBrowse (2)

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

```
void beginBrowse (ResourceType resource,  
                 const char* resName)
```

resource

An enumeration, defined in this class, that indicates the type of resource to be browsed within the CICS system.

resName

The name of the resource that is to be the starting point for browsing the resources.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

dateFormat

Returns the default dateFormat for the CICS region.

```
const char* dateFormat()
```

Conditions

INVREQ

endBrowse

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

```
void endBrowse(ResourceType resource)
```

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

freeStorage

Releases the storage obtained by the **IccSystem** **getStorage** method.

```
void freeStorage(void* pStorage)
```

Conditions

INVREQ

getFile (1)

```
IccFile* getFile(const IccFileId& id)
```

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

getFile (2)

Returns a pointer to the **IccFile** object identified by the argument.

IccFile* `getFile(const char* fileName)`

fileName

The name of a CICS file.

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

getNextFile

This method is only valid after a successful **beginBrowse(IccSystem::file)** call. It returns the next file object in the browse sequence in the CICS system.

IccFile* `getNextFile()`

Conditions

END, FILENOTFOUND, ILLOGIC, NOTAUTH

getStorage

Obtains a block of storage of the requested size and returns a pointer to it. The storage is not released automatically at the end of task; it is only released when a **freeStorage** operation is performed.

void* `getStorage (unsigned long size,
char initByte = -1,
unsigned long storageOpts = 0)`

size

The amount of storage being requested, in bytes

initByte

The initial setting of all bytes in the allocated storage

storageOpts

An enumeration, defined in **IccTask** class, that affects the way that CICS allocates storage.

Conditions

LENGERR, NOSTG

instance

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

static lccSystem* instance()

operatingSystem

char operatingSystem()

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

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

Conditions

NOTAUTH

operatingSystemLevel

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

unsigned short operatingSystemLevel()

Conditions

NOTAUTH

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.

unsigned long release()

Conditions

NOTAUTH

releaseText

Returns the same as **release**, except as a 4-character string. For example, CICS Transaction Server for z/OS [Version 1] Release 3 would return "0130".

const char* releaseText()

Conditions

NOTAUTH

sysId

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

IccSysId& sysId()

Conditions

INVREQ

workArea

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

const IccBuf& workArea()

Conditions

INVREQ

Inherited public methods

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

Requests CICS to abend this task.

```
void abend (const char* abendCode = 0,  
           AbendHandlerOpt opt1 = respectAbendHandler,  
           AbendDumpOpt opt2 = createDump)
```

abendCode

The 4-character abend code

opt1

An enumeration, defined in this class, that indicates whether to respect or ignore any abend handling program specified by **setAbendHandler** method in **IccControl** class

opt2

An enumeration, defined in this class, that indicates whether a dump is to be created.

abendData

```
IccAbendData* abendData()
```

Returns a pointer to an **IccAbendData** object that contains information about the program abends, if any, that relate to this task.

commitUOW

void commitUOW()

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

Conditions

INVREQ, ROLLEDBACK

delay

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

**void delay (const 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.

Conditions

EXPIRED, INVREQ

dump

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

Conditions

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

enterTrace

Writes a user trace entry in the CICS trace table.

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

Conditions

INVREQ, LENGERR

facilityType

Returns an enumeration, defined in this class, that indicates what type of principal facility this task has. This is usually a terminal, such as when the task was started by someone keying a transaction name on a CICS terminal. It is a session if the task is the back end of a mapped APPC conversation.

```
FacilityType facilityType()
```

Conditions

INVREQ

freeStorage

Releases the storage obtained by the **lccTask getStorage** method.

```
void freeStorage(void* pStorage)
```

Conditions

INVREQ

getStorage

Obtains a block of storage of the requested size. The storage is released automatically at the end of task, or when the **freeStorage** operation is performed. See also **getStorage** on page “getStorage” on page 219 in **IccSystem** class.

```
void* getStorage (unsigned long size,  
                 char initByte = -1,  
                 unsigned short storageOpts = 0)
```

size

The amount of storage being requested, in bytes

initByte

The initial setting of all bytes in the allocated storage

storageOpts

An enumeration, defined in this class, that affects the way that CICS allocates storage.

Conditions

LENGERR, NOSTG

instance

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

```
static IccTask* instance();
```

isCommandSecurityOn

```
Icc::Bool isCommandSecurityOn()
```

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

Conditions

INVREQ

isCommitSupported

Returns a boolean, defined in **Icc** structure that indicates whether this task can support the **commit** method. This method returns true in most environments; the exception to this is in a DPL environment (see **link** on page “link” on page 172 in **IccProgram**).

```
Icc::Bool isCommitSupported()
```

Conditions

INVREQ

isResourceSecurityOn

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

Icc::Bool isResourceSecurityOn()

Conditions

INVREQ

isRestarted

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

Icc::Bool isRestarted()

Conditions

INVREQ

isStartDataAvailable

Returns a boolean, defined in **Icc** structure, that indicates whether start data is available for this task. See the **retrieveData** method in **IccStartRequestQ** class if start data is available.

Icc::Bool isStartDataAvailable()

Conditions

INVREQ

number

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

unsigned long number() **const**

principalSysId

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

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

Conditions

INVREQ

priority

Returns the priority for this task.

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

Conditions

INVREQ

rollBackUOW

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

void rollBackUOW()

Conditions

INVREQ, ROLLEDBACK

setDumpOpts

Set the dump options for this task. This method affects the behavior of the **dump** method defined in this class.

void setDumpOpts(unsigned long *opts* = dDefault)

opts

An integer, made by adding or logically ORing values from the **DumpOpts** enumeration, defined in this class.

setPriority

Changes the dispatch priority of this task.

void setPriority(unsigned short *pri*)

pri

The new priority.

Conditions

INVREQ

setWaitText

Sets the text that will appear when someone inquires on this task while it is suspended as a result of a **waitExternal** or **waitOnAlarm** method call.

void setWaitText(const char* *name*)

name

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

startType

StartType startType()

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

Conditions

INVREQ

suspend

Suspend this task, allowing other tasks to be dispatched.

void suspend()

transId

const IccTransId& transId()

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

triggerDataQueueId

const IccDataQueueId& triggerDataQueueId()

Returns a reference to the **IccDataQueueId** representing the trigger queue, if this task was started as a result of data arriving on an **IccDataQueue**. See **startType** method.

Conditions

INVREQ

userId

Returns the ID of the user associated with this task.

const IccUserId& userId(Icc::GetOpt *opt* = Icc::object)

opt

An enumeration, defined in **Icc** structure, that indicates whether the information already existing in the object is to be used or whether it is to be refreshed from CICS.

Conditions

INVREQ

waitExternal

Waits for events that post 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.

```
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.

Conditions

INVREQ

waitOnAlarm

Suspends the task until the alarm goes off (expires). See also **setAlarm** on page “setAlarm” on page 103 in **IccClock**.

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

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

Conditions

INVREQ

workArea

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

```
IccBuf& workArea()
```

Conditions

INVREQ

Inherited public methods

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.

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

A synonym for **empty**. See “Polymorphic Behavior” on page 57 for information on polymorphism.

virtual void clear()

empty

void empty()

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

Conditions

INVREQ, ISCVREQ, NOTAUTH, QIDERR, SYSIDERR

get

A synonym for **readNextItem**. See “Polymorphic Behavior” on page 57 for information on polymorphism.

virtual const IccBuf& get()

numberOfItems

unsigned short numberOfItems() const

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

put

A synonym for **writelnItem**. See “Polymorphic Behavior” on page 57 for information on polymorphism.

virtual void put(const IccBuf& *buffer*)

buffer

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

readItem

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

const IccBuf& readItem(unsigned short *itemNum*)

itemNum

Specifies the item number of the logical record to be retrieved from the queue.

Conditions

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

readNextItem

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

const IccBuf& readNextItem()

Conditions

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

rewriteltem

The parameters are: This method updates the specified item in the temporary storage queue.

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

itemNum

Specifies the item number of the logical record that is to be modified

item

The name of the **IccBuf** object that contains the update data.

opt

An enumeration, defined in this class, that indicates whether the application program is to be suspended if a shortage of space in the queue prevents the record being added. `suspend` is the default.

Conditions

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

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)

This method adds a new record at the end of the temporary storage queue. The returned value is the item number that was created (if this was done successfully).

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

text

The text string that is to added to the end of the temporary storage queue.

opt

An enumeration, defined in this class, that indicates whether the application program is to be suspended if a shortage of space in the queue prevents the record being added. suspend is the default.

Conditions

INVREQ, IOERR, ISCINVREQ, 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 NOSPSPACE 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)

The copy constructor.

IccTempStoreId(const **IccTempStoreId**& *id*)

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

Public methods

operator= (1)

IccTempStoreId& operator=(const char* *name*)

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

operator= (2)

Assigns a new value.

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

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

Inherited public methods

Method
classType

Class
IccBase

Method

className
customClassNum
name
nameLength
operator delete
operator new

Class

IccBase
IccBase
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)

The copy constructor.

IccTermId(const **IccTermId**& *id*)

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

Public methods

operator= (1)

IccTermId& operator=(const char* *name*)

name
The 4-character name of the terminal

operator= (2)

Assigns a new value.

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

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

Inherited public methods

Method
classType

Class
IccBase

Method

className
customClassNum
name
nameLength
operator delete
operator new

Class

IccBase
IccBase
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

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

AIDVal AID()

clear

virtual void clear()

A synonym for **erase**. See “Polymorphic Behavior” on page 57 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

lccTerminalData* data()

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

erase

void erase()

Erase all the data displayed at the terminal.

Conditions

INVREQ, INVPARTN

freeKeyboard

Frees the keyboard so that the terminal can accept input.

void freeKeyboard()

Conditions

INVREQ, INVPARTN

get

A synonym for **receive**. See “Polymorphic Behavior” on page 57 for information on polymorphism.

virtual const lccBuf& get()

height

unsigned short height(lcc::getopt opt = lcc::object)

Returns how many lines the screen holds.

Conditions

INVREQ

inputCursor

Returns the position of the cursor on the screen.

unsigned short inputCursor()

instance

static lccTerminal* instance()

Returns a pointer to the single **lccTerminal** 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)

lccTerminal& operator « (Color color)

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

operator« (2)

lccTerminal& operator « (Highlight highlight)

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

operator« (3)

lccTerminal& operator « (const lccBuf& buffer)

Writes another buffer.

operator« (4)

lccTerminal& operator « (char ch)

Writes a character.

operator« (5)

lccTerminal& operator « (signed char *ch*)

Writes a character.

operator« (6)

lccTerminal& operator « (unsigned char *ch*)

Writes a character.

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)

lccTerminal& operator « (double num)

Writes a double.

operator« (17)

lccTerminal& operator « (long double num)

Writes a long double.

operator« (18)

lccTerminal& operator « (lccTerminal& (*f)(lccTerminal&))

Enables the following syntax:

```
Term « "Hello World" « endl;  
Term « "Hello again" « flush;
```

put

virtual void put(const lccBuf& buf)

A synonym for **sendLine**. See “Polymorphic Behavior” on page 57 for information on polymorphism.

receive

Receives data from the terminal

```
const lccBuf& 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.

Conditions

EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

receive3270Data

Receives the 3270 data buffer from the terminal

```
const lccBuf& 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.

Conditions

INVREQ, LENGERR, TERMERR

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

send (4)

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

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

row

The row where the writing of the data is started.

col

The column where the writing of the data is started.

format

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

... The optional arguments that accompany *format*.

Conditions

INVREQ, LENGERR, TERMERR

send3270Data (1)

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

buffer

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

send3270Data (2)

```
void send3270Data(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 lccBuf& buf)
```

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.

send3270Data (4)

Writes the specified data to either the next line of the terminal or to the specified column of the current line.

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

col

The column where the writing of the data is started

format

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

... The optional arguments that accompany *format*.

Conditions

INVREQ, LENGERR, TERMERR

sendLine (1)

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

buffer

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

sendLine (2)

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

format

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

... The optional arguments that accompany *format*.

sendLine (3)

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

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.

sendLine (4)

Writes the specified data to either the next line of the terminal or to the specified column of the current line.

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

col

The column where the writing of the data is started

format

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

... The optional arguments that accompany *format*.

Conditions

INVREQ, LENGERR, TERMERR

setColor

Changes the color of the text subsequently sent to the terminal.

```
void setColor(Color color=defaultColor)
```

color

An enumeration, defined in this class, that indicates the color of the text that is written to the screen.

setCursor (1)

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

offset

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

setCursor (2)

Two different ways of setting the position of the cursor on the screen.

```
void setCursor (unsigned short row,  
               unsigned short col)
```

row

The row number of the cursor where the top row is 1

col

The column number of the cursor where the left column is 1

Conditions

INVREQ, INVPARTN

setHighlight

Changes the highlighting of the data subsequently sent to the terminal.

```
void setHighlight(Highlight highlight = normal)
```

highlight

An enumeration, defined in this class, that indicates the highlighting of the text that is written to the screen.

setLine

Moves the cursor to the start of line *lineNum*, where 1 is the top line of the terminal. The default is to move the cursor to the start of line 1.

```
void setLine(unsigned short lineNum = 1)
```

lineNum

The line number, counting from the top.

Conditions

INVREQ, INVPARTN

setNewLine

Requests that *numLines* blank lines be sent to the terminal.

```
void setNewLine(unsigned short numLines = 1)
```

numLines

The number of blank lines.

Conditions

INVREQ, INVPARTN

setNextCommArea

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

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

commArea

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

setNextInputMessage

Specifies data that is to be made available, by the **receive** method, to the next transaction started at this terminal.

```
void setNextInputMessage(const lccBuf& message)
```

message

A reference to the buffer that holds the input message.

setNextTransId

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

```
void setNextTransId (const lccTransId& transId,  
                    NextTransIdOpt opt = queue)
```

transId

A reference to the **lccTransId** object that holds the name of a transaction

opt

An enumeration, defined in this class, that indicates whether *transId* should be queued or started immediately (that is, it should be the very next transaction) at this terminal.

signoff

```
void signoff()
```

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

Conditions

INVREQ

signon (1)

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

id A reference to an **lccUserId** object

password

The 8-character existing password.

newPassword

An optional 8-character new password.

signon (2)

Signs the user on to the terminal.

```
void signon (IccUser& user,  
            const char* password = 0,  
            const char* newPassword = 0)
```

user

A reference to an **IccUser** object

password

The 8-character existing password.

newPassword

An optional 8-character new password. This method differs from the first **signon** method in that the **IccUser** object is interrogated to discover **IccGroupId** and language information. The object is also updated with language and ESM return and response codes.

Conditions

INVREQ, NOTAUTH, USERIDERR

waitForAID (1)

AIDVal waitForAID()

Waits for any input and returns an enumeration, defined in this class, that indicates which AID key is expected.

waitForAID (2)

Waits for the specified AID key to be pressed, before returning control. This method loops, receiving input from the terminal, until the correct AID key is pressed by the operator.

```
void waitForAID(AIDVal aid)
```

aid

An enumeration, defined in this class, that indicates which AID key was last pressed.

Conditions

EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

width

Returns the width of the screen in characters.

`unsigned short width(Icc::getopt opt = Icc::object)`

Conditions

INVREQ

workArea

Returns a reference to the **IccBuf** object that holds the terminal work area.

`IccBuf& workArea()`

Inherited public methods

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 “data” on page 246 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

Returns the alternate height of the screen, in lines.

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

opt

An enumeration that indicates whether the information in the object should be refreshed from CICS before being extracted. The default is not to refresh.

Conditions

INVREQ

alternateWidth

Returns the alternate width of the screen, in characters.

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

Conditions

INVREQ

defaultHeight

Returns the default height of the screen, in lines.

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

Conditions

INVREQ

defaultWidth

Returns the default width of the screen, in characters.

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

Conditions

INVREQ

graphicCharCodeSet

Returns the binary code page global identifier as a value in the range 1 to 65534, or 0 for a non-graphics terminal.

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

Conditions

INVREQ

graphicCharSetId

Returns the graphic character set global identifier as a number in the range 1 to 65534, or 0 for a non-graphics terminal.

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

Conditions

INVREQ

isAPLKeyboard

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

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

Conditions

INVREQ

isAPLText

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

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

Conditions

INVREQ

isBTrans

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

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

Conditions

INVREQ

isColor

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

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

Conditions

INVREQ

isEWA

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

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

Conditions

INVREQ

isExtended3270

Returns a Boolean that indicates whether the terminal supports the 3270 extended data stream.

lcc::Bool isExtended3270(lcc::GetOpt *opt* = lcc::object)

Conditions

INVREQ

isFieldOutline

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

`lcc::Bool isFieldOutline(lcc::GetOpt opt = lcc::object)`

Conditions

INVREQ

isGoodMorning

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

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

Conditions

INVREQ

isHighlight

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

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

Conditions

INVREQ

isKatakana

Returns a boolean that indicates whether the terminal supports Katakana.

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

Conditions

INVREQ

isMSRControl

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

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

Conditions

INVREQ

isPS

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

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

Conditions

INVREQ

isSOSI

Returns a boolean that indicates whether the terminal supports mixed EBCDIC/DBCS fields.

`lcc::Bool isSOSI(lcc::GetOpt opt = lcc::object)`

Conditions

INVREQ

isTextKeyboard

Returns a boolean that indicates whether the terminal supports TEXTKYBD.

`lcc::Bool isTextKeyboard(lcc::GetOpt opt = lcc::object)`

Conditions

INVREQ

isTextPrint

Returns a boolean that indicates whether the terminal supports TEXTPRINT.

`lcc::Bool isTextPrint(lcc::GetOpt opt = lcc::object)`

Conditions

INVREQ

isValidation

Returns a boolean that indicates whether the terminal supports validation.

`lcc::Bool isValidation(lcc::GetOpt opt = lcc::object)`

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. lccTime class

lccBase
lccResource
lccTime

lccTime is used to contain time information and is the base class from which **lccAbsTime**, **lccTimeInterval**, and **lccTimeOfDay** classes are derived.

Header file: ICCTIMEH

lccTime constructor (protected)

Constructor

lccTime (**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

Returns the hours component of time—the value specified in the constructor.

virtual unsigned long hours() const

minutes

virtual unsigned long minutes() const

Returns the minutes component of time—the value specified in the constructor.

seconds

virtual unsigned long seconds() const

Returns the seconds component of time—the value specified in the constructor.

timeInHours

virtual unsigned long timeInHours()

Returns the time in hours.

timeInMinutes

virtual unsigned long timeInMinutes()

Returns the time in minutes.

timeInSeconds

virtual unsigned long timeInSeconds()

Returns the time in seconds.

type

Type type() const

Returns an enumeration, defined in this class, that indicates what type of subclass of **IccTime** this is.

Inherited public methods

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

Inherited protected methods

Method	Class
setClassName	IccBase

Method
setCustomClassNum

Class
IccBase

Enumerations

Type

absTime

The object is of **IccAbsTime** class. It is used to represent a current date and time as the number of milliseconds that have elapsed since the beginning of the year 1900.

timeInterval

The object is of **IccTimeInterval** class. It is used to represent a length of time, such as 5 minutes.

timeOfDay

The object is of **IccTimeOfDay** class. It is used to represent a particular time of day, such as midnight.

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)

The copy constructor.

`IccTimeInterval`(`const IccTimeInterval& time`)

Public methods

`operator=`

Assigns one `IccTimeInterval` object to another.

`IccTimeInterval& operator=(const IccTimeInterval& timeInterval)`

`set`

Changes the time held in the `IccTimeInterval` object.

`void set` (`unsigned long hours`,
`unsigned long minutes`,
`unsigned long seconds`)

hours

The new hours setting

minutes
The new minutes setting

seconds
The new seconds setting

Inherited public methods

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)

The copy constructor

`IccTimeOfDay(const IccTimeOfDay& time)`

Public methods

`operator=`

Assigns one `IccTimeOfDay` object to another.

`IccTimeOfDay& operator=(const IccTimeOfDay& timeOfDay)`

`set`

Changes the time held in the `IccTimeOfDay` object.

`void set (unsigned long hours,`
`unsigned long minutes,`
`unsigned long seconds)`

hours

The new hours setting

minutes
The new minutes setting

seconds
The new seconds setting

Inherited public methods

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)

The copy constructor.

IccTPNameId(const IccTPNameId& id)

id A reference to an IccTPNameId object.

Public methods

operator= (1)

IccTPNameId& operator=(const char* name)

name
The 1- to 64-character TP name.

operator= (2)

Assigns a new value.

IccTPNameId& operator=(const IccTPNameId& id)

id A reference to an IccTPNameId object.

Inherited public methods

Method	Class
classType	IccBase
className	IccBase

Method

customClassNum
name
nameLength
operator delete
operator new

Class

IccBase
IccResourceId
IccResourceId
IccBase
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)

The copy constructor.

`IccTransId(const IccTransId& id)`

id A reference to an `IccTransId` object.

Public methods

`operator=` (1)

`IccTransId& operator=(const char* name)`

name
The 4-character transaction name.

`operator=` (2)

Assigns a new value.

`IccTransId& operator=(const IccTransId& id)`

id A reference to an `IccTransId` object.

Inherited public methods

Method
`classType`

Class
`IccBase`

Method

className
customClassNum
name
nameLength
operator delete
operator new

Class

IccBase
IccBase
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 user's group ID.

Constructor (2)

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

userName

The 8-character user ID

gid

The optional 8-character group ID.

Public methods

changePassword

Attempts to change the user's password.

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

password

The user's existing password—a string of up to 8 characters

newPassword

The user's new password—a string of up to 8 characters.

Conditions

INVREQ, NOTAUTH, USERIDERR

daysUntilPasswordExpires

Returns the number of days before the password expires. This method is valid after a successful **verifyPassword** method call in this class.

unsigned short daysUntilPasswordExpires() const

ESMReason

unsigned long ESMReason() const

Returns the external security reason code of interest if a **changePassword** or **verifyPassword** method call is unsuccessful.

ESMResponse

unsigned long ESMResponse() const

Returns the external security response code of interest if a **changePassword** or **verifyPassword** method call is unsuccessful.

groupId

const lccGroupId& groupId() const

Returns a reference to the **lccGroupId** object that holds information on the user's group ID.

invalidPasswordAttempts

unsigned long invalidPasswordAttempts() const

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

language

const char* language() const

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

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

actionOnCondition
actionOnConditionAsChar
actionsOnConditionsText
classType
className
condition
conditionText

Class

IccResource
IccResource
IccResource
IccBase
IccBase
IccResource
IccResource

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

Inherited protected methods

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)

The copy constructor.

IccUserId(const IccUserId& *id*)

id A reference to an IccUserId object.

Public methods

operator= (1)

IccUserId& operator=(const char* *name*)

name
The 8-character name of the user ID.

operator= (2)

Assigns a new value.

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

id A reference to an IccUserId object.

Inherited public methods

Method	Class
classType	IccBase
className	IccBase

Method
customClassNum
name
nameLength
operator delete
operator new

Class
IccBase
IccResourceId
IccResourceId
IccBase
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	EXTENDEDDES	EXTRA	EXTSECURITY
FAILED	FAILEDDBKOUT	FAILINGBKOUT	FCLOSE
FILE	FINALQUIESCE	FINPUT	FIRSTINIT

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	NOOBOPERID	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	OPENERRO

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	RERead	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	SYNCPPOINT	SYNCRECEIVE	SYNCSSEND
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

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 “Storage management” on page 59)
 - Family Subset enforcement (see page “FamilySubset” on page 68)
 - EDF enablement (see page “Execution Diagnostic Facility” on page 48)
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.

- 3** Run the user's application code, using **try** and **catch**, in case the application code does not catch exceptions.
- 4** Create control object.
- 5** Invoke **run** method of control object (defined as pure virtual in **IccControl**).
- 6** Catch any **IccException** objects not caught by the application.
- 7** Call this function to abend task.
- 8** Catch any other exceptions not caught by application.
- 9** Call this function to abend task.
- 10** Return control to CICS.

Part 4. Appendixes

Appendix A. Mapping EXEC CICS calls to Foundation Class methods

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

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	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 SCRNRWD	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	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	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
Note: The retrieveData method gets the start information from CICS and stores it in the IccStartRequestQ object: the information can then be accessed using data , queueName , returnTermId and returnTransId methods.		
RETRIEVE INTO, LENGTH	IccStartRequestQ	data
RETRIEVE QUEUE	IccStartRequestQ	queueName
RETRIEVE RTRANSID	IccStartRequestQ	returnTransId
RETRIEVE RTERMID	IccStartRequestQ	returnTermId
RETURN	IccControl	main ²
Note: Returning (using C++ reserved word return) from method run in class IccControl results in an EXEC CICS RETURN.		
RETURN TRANSID	IccTerminal	setNextTransId ³
RETURN IMMEDIATE	IccTerminal	setNextTransId ³
RETURN COMMAREA LENGTH	IccTerminal	setNextCommArea ³
RETURN INPUTMSG, INPUTMSGLEN	IccTerminal	setNextInputMessage ³
Note: Issue this call before returning from IccControl::run .		
REWRITE	IccFile	rewriteRecord
SEND (APPC)	IccSession	send, sendInvite, sendLast

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	writelnItem
WRITEQ TS	IccTempStore	writelnItem, rewriteItem

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

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	

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

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

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 SCRNHHT
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 SCRNWWD
workArea	ADDRESS TCTUA
IccTerminalData Class	
Method	EXEC CICS
alternateHeight	ASSIGN ALTSCRNHHT
alternateWidth	ASSIGN ALTSCRNWWD
defaultHeight	ASSIGN DEFSCRNHHT
defaultWidth	ASSIGN DEFSCRNWWD
graphicCharSetId	ASSIGN GCHARS
graphicCharCodeSet	ASSIGN GCODES
isAPLKeyboard	ASSIGN APLKYBD
isAPLText	ASSIGN APLTEXT
isBTrans	ASSIGN BTRANS
isColor	ASSIGN COLOR
isEWA	ASSIGN ESASUPP

isExtended3270	ASSIGN EXTDS
isGoodMorning	ASSIGN GMMI
isHighlight	ASSIGN HILIGHT
isKatakana	ASSIGN KATAKANA
isMSRControl	ASSIGN MSRCONTROL
isFieldOutline	ASSIGN OUTLINE
isPS	ASSIGN PS
isSOSI	ASSIGN SOSI
isTextKeyboard	ASSIGN TEXTKYBD
isTextPrint	ASSIGN TEXTPRINT
isValidation	ASSIGN VALIDATION
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$exc3'...
About to read Temporary Storage 'UNKNOWN!'...
IccException: 094 IccTempStore::readNextItem type=CICSCCondition (IccMessage: 008 IccTempStore::readNextItem <CICS returned the 'QIDERR' condition.>)
Program 'icc$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\$RES2 (IRS2)

```
This is program 'icc$res2'...
invoking clear() method for IccDataQueue object
invoking clear() method for IccTempStore object
put() item #1 in IccDataQueue object
put() item #2 in IccDataQueue object
put() item #3 in IccDataQueue object
put() item #1 in IccTempStore object
put() item #2 in IccTempStore object
put() item #3 in IccTempStore object
Now get items from IccDataQueue object
get() from IccDataQueue object returned 'Hello World - item 1'
get() from IccDataQueue object returned 'Hello World - item 2'
get() from IccDataQueue object returned 'Hello World - item 3'
Now get items from IccTempStore object
get() from IccTempStore object returned 'Hello World - item 1'
get() from IccTempStore object returned 'Hello World - item 2'
get() from IccTempStore object returned 'Hello World - item 3'
Program 'icc$res2' complete: Hit PF12 to End
```

ICC\$SEM (ISEM)

```
This is program 'icc$sem'...
Constructing IccSemaphore object (lock by value)...
Issuing lock request... <NORMAL>
Issuing unlock request... <NORMAL>
Constructing Semaphore object (lock by address)...
Issuing tryLock request... <NORMAL>
Issuing unlock request... <NORMAL>
```

```
Program 'icc$sem' complete: Hit PF12 to End
```

ICC\$SES1 (ISE1)

```
This is program 'icc$ses1'...
allocate session... <NORMAL>
STATE=81 ALLOCATED ERR=0 connectProcess...<NORMAL>
STATE=90 SEND ERR=0 sendInvite ... <NORMAL>
STATE=87 PENDRECEIVE ERR=0 receive ... <NORMAL>
STATE=85 FREE ERR=0 - data from back end=[Hi there this is from backEnd
TIME=14:49:18 on 22/02/96]
free... <NORMAL>
STATE=1 NOTAPPLIC ERR=0
```

```
Program 'icc$ses1' 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 rtrn = [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>
- >ICCFEFILE< 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
```

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 2:

Memo to Licensees, GI10-2559
CICS Transaction Server for z/OS Program Directory, GI13-0515
CICS Transaction Server for z/OS Release Guide, GC34-6811
CICS Transaction Server for z/OS Installation Guide, GC34-6812
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, GI13-0515
CICS Transaction Server for z/OS Release Guide, GC34-6811
CICS Transaction Server for z/OS Migration from CICS TS Version 3.1, GC34-6858

CICS Transaction Server for z/OS Migration from CICS TS Version 1.3,
GC34-6855

CICS Transaction Server for z/OS Migration from CICS TS Version 2.2,
GC34-6856

CICS Transaction Server for z/OS Installation Guide, GC34-6812

Administration

CICS System Definition Guide, SC34-6813

CICS Customization Guide, SC34-6814

CICS Resource Definition Guide, SC34-6815

CICS Operations and Utilities Guide, SC34-6816

CICS Supplied Transactions, SC34-6817

Programming

CICS Application Programming Guide, SC34-6818

CICS Application Programming Reference, SC34-6819

CICS System Programming Reference, SC34-6820

CICS Front End Programming Interface User's Guide, SC34-6821

CICS C++ OO Class Libraries, SC34-6822

CICS Distributed Transaction Programming Guide, SC34-6823

CICS Business Transaction Services, SC34-6824

Java Applications in CICS, SC34-6825

JCICS Class Reference, SC34-6001

Diagnosis

CICS Problem Determination Guide, SC34-6826

CICS Messages and Codes, GC34-6827

CICS Diagnosis Reference, GC34-6862

CICS Data Areas, GC34-6863-00

CICS Trace Entries, SC34-6828

CICS Supplementary Data Areas, GC34-6864-00

Communication

CICS Intercommunication Guide, SC34-6829

CICS External Interfaces Guide, SC34-6830

CICS Internet Guide, SC34-6831

Special topics

CICS Recovery and Restart Guide, SC34-6832

CICS Performance Guide, SC34-6833

CICS IMS Database Control Guide, SC34-6834

CICS RACF Security Guide, SC34-6835

CICS Shared Data Tables Guide, SC34-6836

CICS DB2 Guide, SC34-6837

CICS Debugging Tools Interfaces Reference, GC34-6865

CICSplex SM books for CICS Transaction Server for z/OS

General

CICSplex SM Concepts and Planning, SC34-6839

CICSplex SM User Interface Guide, SC34-6840

CICSplex SM Web User Interface Guide, SC34-6841

Administration and Management

CICSplex SM Administration, SC34-6842

CICSplex SM Operations Views Reference, SC34-6843

CICSplex SM Monitor Views Reference, SC34-6844

CICSplex SM Managing Workloads, SC34-6845

CICSplex SM Managing Resource Usage, SC34-6846

CICSplex SM Managing Business Applications, SC34-6847

Programming

CICSplex SM Application Programming Guide, SC34-6848

CICSplex SM Application Programming Reference, SC34-6849

Diagnosis

CICSplex SM Resource Tables Reference, SC34-6850
CICSplex SM Messages and Codes, GC34-6851
CICSplex SM Problem Determination, GC34-6852

CICS family books

Communication

CICS Family: Interproduct Communication, SC34-6853
CICS Family: Communicating from CICS on zSeries, SC34-6854

Licensed publications

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

CICS Diagnosis Reference, GC34-6862
CICS Data Areas, GC34-6863-00
CICS Supplementary Data Areas, GC34-6864-00
CICS Debugging Tools Interfaces Reference, GC34-6865

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 2.

<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 253

Numerics

0 (zero)
in actionOnConditionAsChar 183

A

A
in actionOnConditionAsChar 184
in operatingSystem 220
abend
in lccTask class 223
in Parameter level 55
abend codes 49
abendCode
in lccAbendData class 71
abendCode (parameter)
in abend 223
abendData
in lccTask class 223
AbendDumpOpt
in Enumerations 231
in lccTask class 231
AbendHandlerOpt
in Enumerations 231
in lccTask class 231
abendTask
in ActionOnCondition 188
in CICS conditions 52
absTime
in lccClock class 101
in Type 267
absTime (parameter)
in Constructor 77
in operator= 79
access
in lccFile class 134
Access
in Enumerations 142
in lccFile class 142
access (parameter)
in setAccess 140
Accessing start data
in Starting transactions asynchronously 34
in Using CICS Services 34
accessMethod
in lccFile class 134
action (parameter)
in setActionOnAnyCondition 186
in setActionOnCondition 186
actionOnCondition
in lccResource class 183

ActionOnCondition
in Enumerations 188
in lccResource class 188
actionOnConditionAsChar
in lccResource class 183
actions (parameter)
in setActionsOnConditions 187
actionsOnConditionsText
in lccResource class 184
Activating the trace output
in Debugging Programs 48
in Tracing a Foundation Class Program 48
addable
in Access 142
address
in lccProgram class 171
AID
in lccTerminal class 245
aid (parameter)
in waitForAID 256
AIDVal
in Enumerations 257
in lccTerminal class 257
AIX, CICS for
in Platform differences 54
allocate
in lccSession class 200
AllocateOpt
in Enumerations 208
in lccSession class 208
alternateHeight
in lccTerminalData class 259
in Public methods 259
alternateWidth
in lccTerminalData class 259
in Public methods 259
append
in lccBuf class 92
applName
in lccSystem class 217
ASRAInterrupt
in lccAbendData class 71
in Public methods 71
ASRAKeyType
in lccAbendData class 72
in Public methods 72
ASRAPSW
in lccAbendData class 72
ASRARegisters
in lccAbendData class 72
in Public methods 72
ASRASpaceType
in lccAbendData class 73
in Public methods 73
ASRAStorageType
in lccAbendData class 74
in Public methods 74

- assign
 - in Example of file control 31
 - in lccBuf class 92, 93
 - in lccKey class 161
- automatic
 - in UpdateMode 105
- Automatic condition handling (callHandleEvent)
 - in CICS conditions 52
 - in Conditions, errors, and exceptions 52
- automatic creation 13
- automatic deletion 13
- auxStorage
 - in Location 239

B

- base class
 - overview 15
- Base classes
 - in Overview of the foundation classes 15
- baseName (parameter)
 - in NameOpt 89
- BASESPACE
 - in ASRASpaceType 73
- BDAM 27
- beginBrowse
 - in lccSystem class 217, 218
- beginInsert
 - in Writing records 29
- beginInsert(VSAM only)
 - in lccFile class 134
 - in Public methods 134
- below
 - in StorageOpts 233
- blink
 - in Highlight 258
- blue
 - in Color 258
- Bool
 - in Enumerations 67
 - in lcc structure 67
- BoolSet
 - in Enumerations 67
 - in lcc structure 67
- boolText
 - in Functions 65
 - in lcc structure 65
- browsable
 - in Access 142
- browsing records 30
- Browsing records
 - in File control 30
 - in Using CICS Services 30
- buf (parameter)
 - in dump 224
 - in put 249
 - in send3270Data 252
 - in sendLine 253
 - in setData 211
- buffer
 - in Example of starting transactions 35, 36

- buffer (parameter)
 - in Constructor 92
 - in operator!= 96
 - in operator« 96, 247
 - in operator+= 95
 - in operator= 95
 - in operator== 95
 - in Polymorphic Behavior 58
 - in put 122, 154, 186, 236
 - in registerData 210
 - in rewriteRecord 139
 - in send 250, 251
 - in send3270Data 251, 252
 - in sendLine 252, 253
 - in writeRecord 141
- Buffer objects
 - Data area extensibility 23
 - Data area ownership 23
 - lccBuf constructors 23
 - lccBuf methods 24
 - Working with lccResource subclasses 25
- buffers 23, 26
- byAddress
 - in LockType 197
- byValue
 - in LockType 197

C

- C++ exceptions 49
- C++ Exceptions and the Foundation Classes
 - in Conditions, errors, and exceptions 49
- callHandleEvent
 - in ActionOnCondition 188
 - in CICS conditions 52
- calling conventions 60
- Calling methods on a resource object
 - in Overview of the foundation classes 21
 - in Using CICS resources 21
- callingProgramId
 - in lccControl class 113
 - in Public methods 113
- cancel
 - in Cancelling unexpired start requests 34
 - in lccRequestId class 181
 - in lccStartRequestQ class 209
- cancelAbendHandler
 - in lccControl class 113
- cancelAlarm
 - in lccClock class 101
- Cancelling unexpired start requests
 - in Starting transactions asynchronously 34
 - in Using CICS Services 34
- Case
 - in Enumerations 258
 - in lccTerminal class 258
- caseOpt (parameter)
 - in receive 250
 - in receive3270Data 250

- catch
 - in C++ Exceptions and the Foundation Classes 49, 50
 - in Exception handling (throwException) 53
 - in main function 288
- catchException
 - in Functions 65
 - in lcc structure 65
- CEDF (CICS Execution Diagnostic Facility) 48
- ch (parameter)
 - in operator« 96, 247, 248
- changePassword
 - in lccUser class 277
 - in Public methods 277
- char*
 - in C++ Exceptions and the Foundation Classes 50
- CheckOpt
 - in Enumerations 214
 - in lccStartRequestQ class 214
- CICS
 - in ASRAStorageType 73
 - in GetOpt 68
- CICS conditions
 - abendTask 54
 - automatic condition handling 52
 - Automatic condition handling (callHandleEvent) 52
 - callHandleEvent 52
 - exception handling 53
 - Exception handling (throwException) 53
 - in Conditions, errors, and exceptions 51
 - manual condition handling 52
 - Manual condition handling (noAction) 52
 - noAction 52
 - severe error handling 54
 - Severe error handling (abendTask) 54
 - throwException 53
- CICS Execution Diagnostic Facility (CEDF) 48
- CICS for AIX
 - in Platform differences 54
- CICS OS/2
 - in Platform differences 54
- CICS resources 19
- CICSCondition
 - in C++ Exceptions and the Foundation Classes 51
 - in Type 132
- CICSDataKey
 - in StorageOpts 233
- CICSEXECKEY
 - in ASRAKeyType 72
- CICSInternalTask
 - in StartType 232
- CICSTS13.CICS.SDFHSAMP 6
- CICSTS32.CICS.SDFHC370 6
- CICSTS32.CICS.SDFHLOAD 7
- CICSTS32.CICS.SDFHPROC 6, 7
- CICSTS32.CICS.SDFHSAMP 6
- CICSTS32.CICS.SDFHSDCK 7
- class
 - base 15
 - resource 17
 - resource identification 16
- class (*continued*)
 - singleton 20
 - support 18
- ClassMemoryMgmt
 - in Enumerations 68
 - in lcc structure 68
- className
 - in lccBase class 87
 - in lccEvent class 127
 - in lccException class 130
 - in lccMessage class 167
- className (parameter)
 - in Constructor 129, 167
 - in setClassName 88
- classType
 - in lccBase class 87
 - in lccEvent class 127
 - in lccException class 130
- ClassType
 - in Enumerations 89
 - in lccBase class 89
- classType (parameter)
 - in Constructor 129, 183
- clear
 - in Example of polymorphic behavior 59
 - in lccDataQueue class 121
 - in lccResource class 184
 - in lccTempStore class 236
 - in lccTerminal class 245
 - in Polymorphic Behavior 58
- CLEAR
 - in AIDVal 257
- clearData
 - in lccStartRequestQ class 209
- clearInputMessage
 - in lccProgram class 171
- clearPrefix
 - in lccJournal class 154
- closed
 - in Status 143
- cmmCICS
 - in ClassMemoryMgmt 68
 - in Storage management 60
- cmmDefault
 - in ClassMemoryMgmt 68
 - in Storage management 60
- cmmNonCICS
 - in ClassMemoryMgmt 68
 - in Storage management 60
- CODE/370 48
- Codes
 - in Enumerations 107
 - in lccCondition structure 107
- col (parameter)
 - in send 251
 - in send3270Data 252
 - in sendLine 253
 - in setCursor 254
- Color
 - in Enumerations 258
 - in lccTerminal class 258

- color (parameter)
 - in operator« 247
 - in setColor 253
- commArea
 - in lccControl class 114
- commArea (parameter)
 - in link 172
 - in setNextCommArea 255
- commitOnReturn
 - in CommitOpt 174
- CommitOpt
 - in Enumerations 174
 - in lccProgram class 174
- commitUOW
 - in lccTask class 224
- Compile and link "Hello World"
 - in Hello World 10
- compiling programs 47
- Compiling Programs
 - in Compiling, executing, and debugging 47
- Compiling, executing, and debugging
 - Execution Diagnostic Facility 48
 - Symbolic Debuggers 48
 - Tracing a Foundation Class Program 48
- complete
 - in Kind 163
- complete key 28
- completeLength
 - in lccKey class 161
 - in Public methods 161
- completeLength (parameter)
 - in Constructor 161
- condition
 - in lccEvent class 127
 - in lccResource class 184
 - in Manual condition handling (noAction) 52
 - in Resource classes 17
- condition (parameter)
 - in actionOnCondition 183
 - in actionOnConditionAsChar 183
 - in conditionText 65
 - in setActionOnCondition 186, 187
- condition 0 (NORMAL)
 - in actionsOnConditionsText 184
- condition 1 (ERROR)
 - in actionsOnConditionsText 184
- condition 2 (RDATT)
 - in actionsOnConditionsText 184
- condition 3 (WRBRK)
 - in actionsOnConditionsText 184
- condition 4 (ICCEOF)
 - in actionsOnConditionsText 184
- condition 5 (EODS)
 - in actionsOnConditionsText 184
- condition 6 (EOC)
 - in actionsOnConditionsText 184
- Conditions, errors, and exceptions
 - Automatic condition handling (callHandleEvent) 52
 - Exception handling (throwException) 53
 - Manual condition handling (noAction) 52
 - Method level 55
- Conditions, errors, and exceptions *(continued)*
 - Object level 54
 - Parameter level 55
 - Severe error handling (abendTask) 54
- conditionText
 - in Functions 65
 - in lcc structure 65
 - in lccEvent class 128
 - in lccResource class 185
- ConditionType
 - in Enumerations 189
 - in lccResource class 189
- confirmation
 - in SendOpt 208
- connectProcess
 - in lccSession class 200, 201
 - in Public methods 200, 201
- console
 - in lccControl class 114
- Constructor
 - in lccAbendData class 71
 - in lccAbendData constructor (protected) 71
 - in lccAbsTime class 77
 - in lccAbsTime constructor 77
 - in lccAlarmRequestId class 83
 - in lccAlarmRequestId constructors 83
 - in lccBase class 87
 - in lccBase constructor (protected) 87
 - in lccBuf class 91, 92
 - in lccBuf constructors 91, 92
 - in lccClock class 101
 - in lccClock constructor 101
 - in lccConsole class 109
 - in lccConsole constructor (protected) 109
 - in lccControl class 113
 - in lccControl constructor (protected) 113
 - in lccConvId class 119
 - in lccConvId constructors 119
 - in lccDataQueue class 121
 - in lccDataQueue constructors 121
 - in lccDataQueueId class 125
 - in lccDataQueueId constructors 125
 - in lccEvent class 127
 - in lccEvent constructor 127
 - in lccException class 129
 - in lccException constructor 129
 - in lccFile class 133
 - in lccFile constructors 133
 - in lccFileId class 145
 - in lccFileId constructors 145
 - in lccFileIterator class 147
 - in lccFileIterator constructor 147
 - in lccGroupId class 151
 - in lccGroupId constructors 151
 - in lccJournal class 153
 - in lccJournal constructors 153
 - in lccJournalId class 157
 - in lccJournalId constructors 157
 - in lccJournalTypeld class 159
 - in lccJournalTypeld constructors 159
 - in lccKey class 161

Constructor (*continued*)
 in lccKey constructors 161
 in lccLockId class 165
 in lccLockId constructors 165
 in lccMessage class 167
 in lccMessage constructor 167
 in lccPartnerId class 169
 in lccPartnerId constructors 169
 in lccProgram class 171
 in lccProgram constructors 171
 in lccProgramId class 175
 in lccProgramId constructors 175
 in lccRBA class 177
 in lccRBA constructor 177
 in lccRecordIndex class 179
 in lccRecordIndex constructor (protected) 179
 in lccRequestId class 181
 in lccRequestId constructors 181
 in lccResource class 183
 in lccResource constructor (protected) 183
 in lccResourceId class 191
 in lccResourceId constructors (protected) 191
 in lccRRN class 193
 in lccRRN constructors 193
 in lccSemaphore class 195
 in lccSemaphore constructor 195
 in lccSession class 199, 200
 in lccSession constructor (protected) 200
 in lccSession constructors (public) 199
 in lccStartRequestQ class 209
 in lccStartRequestQ constructor (protected) 209
 in lccSysId class 215
 in lccSysId constructors 215
 in lccSystem class 217
 in lccSystem constructor (protected) 217
 in lccTask class 223
 in lccTask Constructor (protected) 223
 in lccTempStore class 235
 in lccTempStore constructors 235
 in lccTempStoreId class 241
 in lccTempStoreId constructors 241
 in lccTermId class 243
 in lccTermId constructors 243
 in lccTerminal class 245
 in lccTerminal constructor (protected) 245
 in lccTerminalData class 259
 in lccTerminalData constructor (protected) 259
 in lccTime class 265
 in lccTime constructor (protected) 265
 in lccTimeInterval class 269
 in lccTimeInterval constructors 269
 in lccTimeOfDay class 271
 in lccTimeOfDay constructors 271
 in lccTPNameId class 273
 in lccTPNameId constructors 273
 in lccTransId class 275
 in lccTransId constructors 275
 in lccUser class 277
 in lccUser constructors 277
 in lccUserId class 281
 in lccUserId constructors 281

 converse
 in lccSession class 201
 convId
 in lccSession class 201
 convId (parameter)
 in Constructor 119
 convName (parameter)
 in Constructor 119
 in operator= 119
 copt (parameter)
 in setStartOpts 212, 213
 createDump
 in AbendDumpOpt 231
 creating a resource object 19
 Creating a resource object
 in Overview of the foundation classes 19
 in Using CICS resources 19
 Singleton classes 20
 Creating an object
 in C++ Objects 13
 creating object 13
 current (parameter)
 in setPrefix 154
 cursor
 in Finding out information about a terminal 42
 in lccTerminal class 245
 customClassNum
 in lccBase class 88
 in Public methods 88
 cut
 in lccBuf class 93
 in lccBuf constructors 24
 CVDA
 in Enumeration 283
 in lccValue structure 283
 cyan
 in Color 258

D
 data
 in Accessing start data 34
 in Finding out information about a terminal 42
 in lccStartRequestQ class 210
 in lccTerminal class 246
 in lccTerminalData class 259
 data (parameter)
 in enterTrace 225
 in put 204
 data area extensibility 23
 Data area extensibility
 in Buffer objects 23
 in lccBuf class 23
 data area ownership 23
 Data area ownership
 in Buffer objects 23
 in lccBuf class 23
 dataArea
 in lccBuf class 93
 dataArea (parameter)
 in append 92

- dataArea (parameter) *(continued)*
 - in assign 93, 161
 - in Constructor 91
 - in insert 94
 - in overlay 98
 - in replace 98
- dataAreaLength
 - in lccBuf class 93
 - in Public methods 93
- dataAreaOwner
 - in Data area ownership 23
 - in lccBuf class 93
- DataAreaOwner
 - in Enumerations 99
 - in lccBuf class 99
- dataAreaType
 - in Data area extensibility 23
 - in lccBuf class 94
- DataAreaType
 - in Enumerations 99
 - in lccBuf class 99
- dataItems
 - in Example of polymorphic behavior 58
- dataLength
 - in lccBuf class 94
- dataqueue
 - in FacilityType 232
- dataQueueTrigger
 - in StartType 232
- date
 - in lccAbsTime class 77
 - in lccClock class 102
- date services 44
- dateFormat
 - in lccSystem class 218
- DateFormat
 - in Enumerations 104
 - in lccClock class 104
- dateSeparator (parameter)
 - in date 77, 102
 - in Example of time and date services 44
- dayOfMonth
 - in Example of time and date services 45
 - in lccAbsTime class 78
 - in lccClock class 102
- dayOfWeek
 - in Example of time and date services 45
 - in lccAbsTime class 78
 - in lccClock class 102
- DayOfWeek
 - in Enumerations 105
 - in lccClock class 105
- daysSince1900
 - in Example of time and date services 45
 - in lccAbsTime class 78
 - in lccClock class 102
- daysUntilPasswordExpires
 - in lccUser class 278
- dComplete
 - in DumpOpts 232
- dDCT
 - in DumpOpts 232
- dDefault
 - in DumpOpts 232
- debuggers 48
- debugging programs 47
- Debugging Programs
 - Activating the trace output 48
 - Enabling EDF 48
 - Execution Diagnostic Facility 48
 - in Compiling, executing, and debugging 47
 - Symbolic Debuggers 48
 - Tracing a Foundation Class Program 48
- defaultColor
 - in Color 258
- defaultHeight
 - in lccTerminalData class 260
 - in Public methods 260
- defaultHighlight
 - in Highlight 258
- defaultWidth
 - in lccTerminalData class 260
 - in Public methods 260
- delay
 - in lccTask class 224
 - in Support Classes 19
- deletable
 - in Access 143
- delete
 - in Deleting an object 14
 - in Storage management 60
- delete operator 13
- deleteLockedRecord 30
 - in Deleting locked records 30
 - in lccFile class 134
- deleteRecord
 - in Deleting normal records 30
 - in lccFile class 135
- deleteRecord method 30
- Deleting an object
 - in C++ Objects 14
- deleting items 40
- Deleting items
 - in Temporary storage 40
 - in Using CICS Services 40
- Deleting locked records
 - in Deleting records 30
 - in File control 30
- Deleting normal records
 - in Deleting records 30
 - in File control 30
- deleting queues 38
- Deleting queues
 - in Transient Data 38
 - in Using CICS Services 38
- deleting records 30
- Deleting records
 - Deleting locked records 30
 - Deleting normal records 30
 - in File control 30
 - in Using CICS Services 30

- dFCT
 - in DumpOpts 232
- DFHCURDI 7
- DFHCURDS 6, 7
- disabled
 - in Status 143
- doSomething
 - in Using an object 14
- dPCT
 - in DumpOpts 232
- DPL
 - in StartType 232
- dPPT
 - in DumpOpts 232
- dProgram
 - in DumpOpts 232
- dSIT
 - in DumpOpts 232
- dStorage
 - in DumpOpts 232
- dTables
 - in DumpOpts 232
- dTask
 - in DumpOpts 232
- dTCT
 - in DumpOpts 232
- dTerminal
 - in DumpOpts 232
- dTRT
 - in DumpOpts 232
- dump
 - in lccTask class 224
- dumpCode (parameter)
 - in dump 224
- DumpOpts
 - in Enumerations 231
 - in lccTask class 231
- dynamic creation 13
- dynamic deletion 13
- dynamic link library 6
- Dynamic link library
 - in Installed contents 6
 - Location 6

E

- ECBList (parameter)
 - in waitExternal 230
- EDF (Execution Diagnostic Facility) 48
- EDF (parameter)
 - in initializeEnvironment 66
- empty
 - in Deleting items 40
 - in Deleting queues 38
 - in lccDataQueue class 121
 - in lccTempStore class 236
 - in Temporary storage 39
 - in Transient Data 37
- enabled
 - in Status 143

- enableStatus
 - in lccFile class 135
- Enabling EDF
 - in Debugging Programs 48
 - in Execution Diagnostic Facility 48
- endBrowse
 - in lccSystem class 218
- endInsert
 - in Writing records 29
- endInsert(VSAM only)
 - in lccFile class 135
 - in Public methods 135
- endl
 - in Example of terminal control 43
- ENTER
 - in AIDVal 257
- enterTrace
 - in lccTask class 225
- entryPoint
 - in lccProgram class 172
- Enumeration
 - CVDA 283
 - in lccValue structure 283
- Enumerations
 - AbendDumpOpt 231
 - AbendHandlerOpt 231
 - Access 142
 - ActionOnCondition 188
 - AIDVal 257
 - AllocateOpt 208
 - Bool 67
 - BoolSet 67
 - Case 258
 - CheckOpt 214
 - ClassMemoryMgmt 68
 - ClassType 89
 - Codes 107
 - Color 258
 - CommitOpt 174
 - ConditionType 189
 - DataAreaOwner 99
 - DataAreaType 99
 - DateFormat 104
 - DayOfWeek 105
 - DumpOpts 231
 - FacilityType 232
 - FamilySubset 68
 - GetOpt 68
 - HandleEventReturnOpt 188
 - Highlight 258
 - in lcc structure 67
 - in lccBase class 89
 - in lccBuf class 99
 - in lccClock class 104
 - in lccCondition structure 107
 - in lccConsole class 112
 - in lccException class 131
 - in lccFile class 142
 - in lccJournal class 156
 - in lccKey class 163
 - in lccProgram class 174

- Enumerations (*continued*)
 - in `IccRecordIndex` class 180
 - in `IccResource` class 188
 - in `IccSemaphore` class 197
 - in `IccSession` class 208
 - in `IccStartRequestQ` class 214
 - in `IccSystem` class 222
 - in `IccTask` class 231
 - in `IccTempStore` class 239
 - in `IccTerminal` class 257
 - in `IccTime` class 267
 - Kind 163
 - LifeTime 197
 - LoadOpt 174
 - Location 239
 - LockType 197
 - MonthOfYear 105
 - NameOpt 89
 - NextTransIdOpt 258
 - NoSpaceOpt 239
 - Options 156
 - Platforms 69
 - ProtectOpt 214
 - Range 107
 - ReadMode 143
 - ResourceType 222
 - RetrieveOpt 214
 - SearchCriterion 143
 - SendOpt 208
 - SeverityOpt 112
 - StartType 232
 - StateOpt 208
 - Status 143
 - StorageOpts 232
 - SyncLevel 208
 - TraceOpt 233
 - Type 131, 180, 267
 - UpdateMode 105
 - WaitPostType 233
 - WaitPurgeability 233
- `equalToKey`
 - in `SearchCriterion` 143
- `erase`
 - in Example of terminal control 43
 - in Hello World 9
 - in `IccTerminal` class 246
 - in Sending data to a terminal 42
- `errorCode`
 - in `IccSession` class 201
- ESDS
 - in File control 27
- ESDS file 27
- ESMReason
 - in `IccUser` class 278
- ESMResponse
 - in `IccUser` class 278
- event (parameter)
 - in `handleEvent` 185
- Example of file control
 - in File control 30
 - in Using CICS Services 30
- Example of managing transient data
 - in Transient Data 38
 - in Using CICS Services 38
- Example of polymorphic behavior
 - in Miscellaneous 58
 - in Polymorphic Behavior 58
- Example of starting transactions
 - in Starting transactions asynchronously 35
 - in Using CICS Services 35
- Example of Temporary Storage
 - in Temporary storage 40
 - in Using CICS Services 40
- Example of terminal control
 - in Terminal control 42
 - in Using CICS Services 42
- Example of time and date services
 - in Time and date services 44
 - in Using CICS Services 44
- exception
 - in `TraceOpt` 233
- exception (parameter)
 - in `catchException` 65
- Exception handling (`throwException`)
 - in CICS conditions 53
 - in Conditions, errors, and exceptions 53
- `exceptionNum` (parameter)
 - in Constructor 129
- exceptions 49
- `exceptionType` (parameter)
 - in Constructor 129
- Executing Programs
 - in Compiling, executing, and debugging 47
- Execution Diagnostic Facility
 - Enabling EDF 48
 - in Compiling, executing, and debugging 48
 - in Debugging Programs 48
- Execution Diagnostic Facility (EDF) 48
- Expected Output from "Hello World"
 - in Hello World 10
 - in Running "Hello World" on your CICS server 10
- extensible
 - in `DataAreaType` 99
- external
 - in `DataAreaOwner` 99
- `extractProcess`
 - in `IccSession` class 202
- `extractState`
 - in `StateOpt` 208

F

- `facilityType`
 - in `IccTask` class 225
- FacilityType
 - in Enumerations 232
 - in `IccTask` class 232
- `fam` (parameter)
 - in `initializeEnvironment` 66
- `familyConformanceError`
 - in C++ Exceptions and the Foundation Classes 51
 - in Type 132

- FamilySubset
 - in Enumerations 68
 - in lcc structure 68
- FEPIRequest
 - in StartType 232
- file (parameter)
 - in Constructor 147
 - in Example of file control 31
- file control
 - browsing records 30
 - deleting records 30
 - example 30
 - rewriting records 29
 - updating records 29
- File control
 - Browsing records 30
 - Deleting locked records 30
 - Deleting normal records 30
 - Deleting records 30
 - Example of file control 30
 - in Using CICS Services 27
 - Reading ESDS records 28
 - Reading KSDS records 28
 - Reading records 27
 - Reading RRDS records 28
 - Updating records 29
 - Writing ESDS records 29
 - Writing KSDS records 29
 - Writing records 28
 - Writing RRDS records 29
- fileName (parameter)
 - in Constructor 133, 145
 - in getFile 219
 - in operator= 145
- Finding out information about a terminal
 - in Terminal control 42
 - in Using CICS Services 42
- First Screen
 - in ICC\$PRG1 (IPR1) 305
 - in Output from sample programs 305
- fixed
 - in DataAreaType 99
- flush
 - in Example of terminal control 43
 - in lccSession class 202
- for
 - in Example of file control 31
- Form
 - in Polymorphic Behavior 58
- format (parameter)
 - in append 92
 - in assign 93
 - in date 77, 102
 - in Example of time and date services 44
 - in send 250, 251
 - in send3270Data 251, 252
 - in sendLine 252, 253
- Foundation Class Abend codes
 - in Conditions, errors, and exceptions 49
- free
 - in lccSession class 202

- freeKeyboard
 - in lccTerminal class 246
 - in Sending data to a terminal 42
- freeStorage
 - in lccSystem class 218
 - in lccTask class 225
- fsAllowPlatformVariance
 - in FamilySubset 68
 - in Platform differences 54
- fsDefault
 - in FamilySubset 68
- fsEnforce
 - in FamilySubset 68
 - in Platform differences 54
- fullAccess
 - in Access 143
- Functions
 - boolText 65
 - catchException 65
 - conditionText 65
 - in lcc structure 65
 - initializeEnvironment 66
 - isClassMemoryMgmtOn 66
 - isEDFOn 66
 - isFamilySubsetEnforcementOn 66
 - returnToCICS 66
 - setEDF 67
 - unknownException 67

G

- generic
 - in Kind 163
- generic key 28
- get
 - in Example of polymorphic behavior 59
 - in lccDataQueue class 122
 - in lccResource class 185
 - in lccSession class 202
 - in lccTempStore class 236
 - in lccTerminal class 246
 - in Polymorphic Behavior 58
- getFile
 - in lccSystem class 218, 219
- getNextFile
 - in lccSystem class 219
- GetOpt
 - in Enumerations 68
 - in lcc structure 68
- getStorage
 - in lccSystem class 219
 - in lccTask class 226
- gid (parameter)
 - in Constructor 277
- graphicCharCodeSet
 - in lccTerminalData class 260
- graphicCharSetId
 - in lccTerminalData class 260
- green
 - in Color 258

- groupid
 - in IccUser class 278
- groupName (parameter)
 - in Constructor 151, 277
 - in operator= 151
- gteqToKey
 - in SearchCriterion 143

H

- H
 - in actionOnConditionAsChar 184
- handleEvent
 - in Automatic condition handling (callHandleEvent) 52, 53
 - in IccResource class 185
- HandleEventReturnOpt
 - in Enumerations 188
 - in IccResource class 188
- handPost
 - in WaitPostType 233
- Header files
 - in Installed contents 5
 - Location 6
- height
 - in IccTerminal class 246
- Hello World
 - commentary 9
 - Compile and link 10
 - Expected Output from "Hello World" 10
 - running 10
- Highlight
 - in Enumerations 258
 - in IccTerminal class 258
- highlight (parameter)
 - in operator« 247
 - in setHighlight 254
- hold
 - in LoadOpt 174
- hours
 - in IccAbsTime class 78
 - in IccTime class 265
- hours (parameter)
 - in Constructor 265, 269, 271
 - in set 269, 271

I

- Icc
 - in Method level 55
 - in Overview of the foundation classes 15
- Icc structure
 - Bool 67
 - BoolSet 67
 - boolText 65
 - catchException 65
 - ClassMemoryMgmt 68
 - conditionText 65
 - FamilySubset 68
 - GetOpt 68
 - initializeEnvironment 66

- Icc structure (*continued*)
 - isClassMemoryMgmtOn 66
 - isEDFOn 66
 - isFamilySubsetEnforcementOn 66
 - Platforms 69
 - returnToCICS 66
 - setEDF 67
 - unknownException 67
- Icc::initializeEnvironment
 - in Storage management 60
- ICC\$BUF 6
- ICC\$BUF (IBUF)
 - in Output from sample programs 303
- ICC\$CLK 6
- ICC\$CLK (ICLK)
 - in Output from sample programs 303
- ICC\$DAT (IDAT)
 - in Output from sample programs 303
- ICC\$EXC1 (IEX1)
 - in Output from sample programs 304
- ICC\$EXC2 (IEX2)
 - in Output from sample programs 304
- ICC\$EXC3 (IEX3)
 - in Output from sample programs 304
- ICC\$FIL (IFIL)
 - in Output from sample programs 304
- ICC\$HEL 6
- ICC\$HEL (IHEL)
 - in Output from sample programs 304
- ICC\$JRN (IJRN)
 - in Output from sample programs 305
- ICC\$PRG1 (IPR1)
 - First Screen 305
 - in Output from sample programs 305
 - Second Screen 305
- ICC\$RES1 (IRS1)
 - in Output from sample programs 305
- ICC\$RES2 (IRS2)
 - in Output from sample programs 306
- ICC\$SEM (ISEM)
 - in Output from sample programs 306
- ICC\$SES1 6
- ICC\$SES1 (ISE1)
 - in Output from sample programs 306
- ICC\$SES2 6
 - in Output from sample programs 307
- ICC\$SRQ1 (ISR1)
 - in Output from sample programs 307
- ICC\$SRQ2 (ISR2)
 - in Output from sample programs 307
- ICC\$SYS (ISYS)
 - in Output from sample programs 308
- ICC\$TMP (ITMP)
 - in Output from sample programs 308
- ICC\$TRM (ITRM)
 - in Output from sample programs 308
- ICC\$TSK (ITSK)
 - in Output from sample programs 309
- IccAbendData
 - in Singleton classes 20

- lccAbendData class
 - abendCode 71
 - ASRAInterrupt 71
 - ASRAKeyType 72
 - ASRAPSW 72
 - ASRARegisters 72
 - ASRASpaceType 73
 - ASRAStorageType 74
 - Constructor 71
 - instance 74
 - isDumpAvailable 74
 - originalAbendCode 74
 - programName 74
- lccAbendData constructor (protected)
 - Constructor 71
 - in lccAbendData class 71
- lccAbsTime
 - in Base classes 16
 - in delay 224
 - in lccTime class 265
 - in Support Classes 19
 - in Time and date services 44
- lccAbsTime class
 - Constructor 77
 - date 77
 - dayOfMonth 78
 - dayOfWeek 78
 - daysSince1900 78
 - hours 78
 - milliSeconds 78
 - minutes 78
 - monthOfYear 79
 - operator= 79
 - packedDecimal 79
 - seconds 79
 - time 79
 - timeInHours 79
 - timeInMinutes 80
 - timeInSeconds 80
 - year 80
- lccAbsTime constructor
 - Constructor 77
 - in lccAbsTime class 77
- lccAbsTime,
 - in Support Classes 19
- lccAlarmRequestId
 - in lccAlarmRequestId class 83
- lccAlarmRequestId class
 - Constructor 83
 - isExpired 84
 - operator= 84
 - setTimerECA 84
 - timerECA 84
- lccAlarmRequestId constructors
 - Constructor 83
 - in lccAlarmRequestId class 83
- lccBase
 - in Base classes 15
 - in lccAbendData class 71
 - in lccAbsTime class 77
 - in lccAlarmRequestId class 83
- lccBase (*continued*)
 - in lccBase class 87
 - in lccBuf class 91
 - in lccClock class 101
 - in lccConsole class 109
 - in lccControl class 113
 - in lccConvId class 119
 - in lccDataQueue class 121
 - in lccDataQueueId class 125
 - in lccEvent class 127
 - in lccException class 129
 - in lccFile class 133
 - in lccFileId class 145
 - in lccFileIterator class 147
 - in lccGroupId class 151
 - in lccJournal class 153
 - in lccJournalId class 157
 - in lccJournalTypeId class 159
 - in lccKey class 161
 - in lccLockId class 165
 - in lccMessage class 167
 - in lccPartnerId class 169
 - in lccProgram class 171
 - in lccProgramId class 175
 - in lccRBA class 177
 - in lccRecordIndex class 179
 - in lccRequestId class 181
 - in lccResource class 183
 - in lccResourceId class 191
 - in lccRRN class 193
 - in lccSemaphore class 195
 - in lccSession class 199
 - in lccStartRequestQ class 209
 - in lccSysId class 215
 - in lccSystem class 217
 - in lccTask class 223
 - in lccTempStore class 235
 - in lccTempStoreId class 241
 - in lccTermId class 243
 - in lccTerminal class 245
 - in lccTerminalData class 259
 - in lccTime class 265
 - in lccTimeInterval class 269
 - in lccTimeOfDay class 271
 - in lccTPNameId class 273
 - in lccTransId class 275
 - in lccUser class 277
 - in lccUserId class 281
 - in Resource classes 17
 - in Resource identification classes 16
 - in Storage management 60
 - in Support Classes 18
- lccBase class
 - className 87
 - classType 87
 - ClassType 89
 - Constructor 87
 - customClassNum 88
 - NameOpt 89
 - operator delete 88
 - operator new 88

- lccBase class *(continued)*
 - overview 15
 - setClassName 88
 - setCustomClassNum 88
- lccBase constructor (protected)
 - Constructor 87
 - in lccBase class 87
- lccBuf
 - in Buffer objects 23
 - in C++ Exceptions and the Foundation Classes 51
 - in Data area extensibility 23
 - in Data area ownership 23
 - in Example of file control 31
 - in Example of managing transient data 39
 - in Example of polymorphic behavior 59
 - in Example of starting transactions 35, 36, 37
 - in Example of Temporary Storage 41
 - in Example of terminal control 43
 - in lccBuf class 23, 91
 - in lccBuf constructors 23, 24
 - in lccBuf methods 24
 - in Reading data 38
 - in Reading items 40
 - in Scope of data in lccBuf reference returned from 'read' methods 61
 - in Support Classes 19
 - in Working with lccResource subclasses 25, 26
- lccBuf class
 - append 92
 - assign 92, 93
 - Constructor 91, 92
 - constructors 23
 - cut 93
 - data area extensibility 23
 - Data area extensibility 23
 - data area ownership 23
 - Data area ownership 23
 - dataArea 93
 - dataAreaLength 93
 - dataAreaOwner 93
 - DataAreaOwner 99
 - dataAreaType 94
 - DataAreaType 99
 - dataLength 94
 - lccBuf constructors 23
 - lccBuf methods 24
 - in Buffer objects 23
 - insert 94
 - isFMHContained 94
 - methods 24
 - operator const char* 94
 - operator!= 96
 - operator<< 96, 98
 - operator+= 95
 - operator= 95
 - operator== 95
 - overlay 98
 - replace 98
 - setDataLength 98
 - setFMHContained 99
 - Working with lccResource subclasses 25
- lccBuf constructors 23
 - Constructor 91, 92
 - in Buffer objects 23
 - in lccBuf class 23, 91
- lccBuf methods 24
 - in Buffer objects 24
 - in lccBuf class 24
- lccBuf reference 61
- lccClock
 - in Example of time and date services 44, 45
 - in lccAlarmRequestId class 83
 - in lccClock class 101
 - in Time and date services 44
- lccClock class
 - absTime 101
 - cancelAlarm 101
 - Constructor 101
 - date 102
 - DateFormat 104
 - dayOfMonth 102
 - dayOfWeek 102
 - DayOfWeek 105
 - daysSince1900 102
 - milliseconds 103
 - monthOfYear 103
 - MonthOfYear 105
 - setAlarm 103
 - time 103
 - update 104
 - UpdateMode 105
 - year 104
- lccClock constructor
 - Constructor 101
 - in lccClock class 101
- lccCondition
 - in C++ Exceptions and the Foundation Classes 51
- lccCondition structure
 - Codes 107
 - Range 107
- lccConsole
 - in Buffer objects 23
 - in Object level 54, 55
 - in Singleton classes 20
- lccConsole class
 - Constructor 109
 - instance 109
 - overview 20
 - put 109
 - replyTimeout 109
 - resetRouteCodes 110
 - setAllRouteCodes 110
 - setReplyTimeout 110
 - setRouteCodes 110
 - SeverityOpt 112
 - write 110
 - writeAndGetReply 111
- lccConsole constructor (protected)
 - Constructor 109
 - in lccConsole class 109
- lccControl
 - in Base classes 15

- lccControl *(continued)*
 - in Example of starting transactions 36
 - in Hello World 9
 - in lccControl class 113
 - in lccProgram class 171
 - in main function 287, 288
 - in Mapping EXEC CICS calls to Foundation Class methods 291
 - in Method level 55
 - in Singleton classes 20
 - in Support Classes 19
- lccControl class
 - callingProgramId 113
 - cancelAbendHandler 113
 - commArea 114
 - console 114
 - Constructor 113
 - initData 114
 - instance 114
 - isCreated 114
 - overview 15, 20
 - programId 114
 - resetAbendHandler 115
 - returnProgramId 115
 - run 115
 - session 115
 - setAbendHandler 115
 - startRequestQ 116
 - system 116
 - task 116
 - terminal 116
- lccControl constructor (protected)
 - Constructor 113
 - in lccControl class 113
- lccControl::run
 - in Mapping EXEC CICS calls to Foundation Class methods 291
- lccConvId
 - in lccConvId class 119
- lccConvId class
 - Constructor 119
 - operator= 119
- lccConvId constructors
 - Constructor 119
 - in lccConvId class 119
- lccDataQueue
 - in Buffer objects 23
 - in Example of managing transient data 38, 39
 - in Example of polymorphic behavior 59
 - in Resource classes 17
 - in Temporary storage 39
 - in Transient Data 37, 38
 - in Working with lccResource subclasses 25
 - in Writing data 38
- lccDataQueue class
 - clear 121
 - Constructor 121
 - empty 121
 - get 122
 - put 122
 - readItem 122
- lccDataQueue class *(continued)*
 - writelnItem 122
- lccDataQueue constructors
 - Constructor 121
 - in lccDataQueue class 121
- lccDataQueueId
 - in Example of managing transient data 38
 - in lccDataQueueId class 125
 - in Transient Data 37, 38
- lccDataQueueId class
 - Constructor 125
 - operator= 125
- lccDataQueueId constructors
 - Constructor 125
 - in lccDataQueueId class 125
- lccEvent
 - in lccEvent class 127
 - in Support Classes 19
- lccEvent class
 - className 127
 - classType 127
 - condition 127
 - conditionText 128
 - Constructor 127
 - methodName 128
 - summary 128
- lccEvent constructor
 - Constructor 127
 - in lccEvent class 127
- lccException
 - in C++ Exceptions and the Foundation Classes 50, 51
 - in lccException class 129
 - in lccMessage class 167
 - in main function 288
 - in Method level 55
 - in Object level 55
 - in Parameter level 55, 56
 - in Support Classes 19
- lccException class
 - CICSCondition type 51
 - className 130
 - classType 130
 - Constructor 129
 - familyConformanceError type 51
 - internalError type 51
 - invalidArgument type 50
 - invalidMethodCall type 51
 - message 130
 - methodName 130
 - number 130
 - objectCreationError type 50
 - summary 130
 - type 131
 - Type 131
 - typeText 131
- lccException constructor
 - Constructor 129
 - in lccException class 129
- ICCFCC 7
- ICCFCL 6, 7

- ICCFCDLL 6
- ICCFCGL 7
- ICCFCIMP 7
- ICCFCL 7
- lccFile
 - in Browsing records 30
 - in Buffer objects 23
 - in C++ Exceptions and the Foundation Classes 51
 - in Deleting locked records 30
 - in Deleting normal records 30
 - in Example of file control 30
 - in File control 27
 - in lccFile class 133
 - in lccFileIterator class 147
 - in Reading ESDS records 28
 - in Reading KSDS records 28
 - in Reading records 27, 28
 - in Reading RRDS records 28
 - in Resource identification classes 16
 - in Singleton classes 20
 - in Updating records 29
 - in Writing ESDS records 29
 - in Writing KSDS records 29
 - in Writing records 28, 29
 - in Writing RRDS records 29
- lccFile class
 - access 134
 - Access 142
 - accessMethod 134
 - beginInsert(VSAM only) 134
 - Constructor 133
 - deleteLockedRecord 30, 134
 - deleteRecord 135
 - deleteRecord method 30
 - enableStatus 135
 - endInsert(VSAM only) 135
 - isAddable 135
 - isBrowsable 136
 - isDeletable 136
 - isEmptyOnOpen 136
 - isReadable 136
 - isReadable method 28
 - isRecoverable 137
 - isUpdatable 137
 - keyLength 137
 - keyLength method 28
 - keyPosition 137
 - keyPosition method 28
 - openStatus 138
 - ReadMode 143
 - readRecord 138
 - readRecord method 28
 - recordFormat 138
 - recordFormat method 28
 - recordIndex 139
 - recordIndex method 28
 - recordLength 139
 - recordLength method 28
 - registerRecordIndex 28, 139
 - registerRecordIndex method 28
 - rewriteRecord 139
- lccFile class (*continued*)
 - rewriteRecord method 29
 - SearchCriterion 143
 - setAccess 140
 - setEmptyOnOpen 140
 - setStatus 140
 - Status 143
 - type 141
 - unlockRecord 141
 - writeRecord 141
 - writeRecord method 28
- lccFile constructors
 - Constructor 133
 - in lccFile class 133
- lccFile::readRecord
 - in Scope of data in lccBuf reference returned from 'read' methods 61
- lccFileId
 - in Base classes 15
 - in File control 27
 - in lccFileId class 145
 - in Resource identification classes 16
- lccFileId class
 - Constructor 145
 - operator= 145
 - overview 15, 27
 - reading records 27
- lccFileId constructors
 - Constructor 145
 - in lccFileId class 145
- lccFileIterator
 - in Browsing records 30
 - in Buffer objects 23
 - in Example of file control 30, 31
 - in File control 27
 - in lccFileIterator class 147
- lccFileIterator class
 - Constructor 147
 - overview 27
 - readNextRecord 147
 - readNextRecord method 30
 - readPreviousRecord 30, 148
 - reset 148
- lccFileIterator constructor
 - Constructor 147
 - in lccFileIterator class 147
- lccGroupId
 - in lccGroupId class 151
- lccGroupId class
 - Constructor 151
 - operator= 151
- lccGroupId constructors
 - Constructor 151
 - in lccGroupId class 151
- lccJournal
 - in Buffer objects 23
 - in lccJournal class 153
 - in Object level 54, 55
- lccJournal class
 - clearPrefix 154
 - Constructor 153

- lccJournal class *(continued)*
 - journalTypeId 154
 - Options 156
 - put 154
 - registerPrefix 154
 - setJournalTypeId 154
 - setPrefix 154
 - wait 155
 - writeRecord 155
- lccJournal constructors
 - Constructor 153
 - in lccJournal class 153
- lccJournalId
 - in lccJournalId class 157
- lccJournalId class
 - Constructor 157
 - number 157
 - operator= 157
- lccJournalId constructors
 - Constructor 157
 - in lccJournalId class 157
- lccJournalTypeId
 - in lccJournalTypeId class 159
- lccJournalTypeId class
 - Constructor 159
 - operator= 159
- lccJournalTypeId constructors
 - Constructor 159
 - in lccJournalTypeId class 159
- lccKey
 - in Browsing records 30
 - in Deleting normal records 30
 - in File control 27
 - in lccKey class 161
 - in lccRecordIndex class 179
 - in Reading KSDS records 28
 - in Reading records 27
 - in Writing KSDS records 29
 - in Writing records 28
- lccKey class 28
 - assign 161
 - completeLength 161
 - Constructor 161
 - kind 162
 - Kind 163
 - operator!= 162, 163
 - operator= 162
 - operator== 162
 - reading records 27
 - setKind 163
 - value 163
- lccKey constructors
 - Constructor 161
 - in lccKey class 161
- lccLockId
 - in lccLockId class 165
- lccLockId class
 - Constructor 165
 - operator= 165
- lccLockId constructors
 - Constructor 165

- lccLockId constructors *(continued)*
 - in lccLockId class 165
- lccMessage
 - in lccMessage class 167
 - in Support Classes 19
- lccMessage class
 - className 167
 - Constructor 167
 - methodName 167
 - number 167
 - summary 168
 - text 168
- lccMessage constructor
 - Constructor 167
 - in lccMessage class 167
- lccPartnerId
 - in lccPartnerId class 169
- lccPartnerId class
 - Constructor 169
 - operator= 169
- lccPartnerId constructors
 - Constructor 169
 - in lccPartnerId class 169
- lccProgram
 - in Buffer objects 23
 - in lccProgram class 171
 - in Program control 32, 33
 - in Resource classes 17
- lccProgram class
 - address 171
 - clearInputMessage 171
 - CommitOpt 174
 - Constructor 171
 - entryPoint 172
 - length 172
 - link 172
 - load 173
 - LoadOpt 174
 - program control 32
 - setInputMessage 173
 - unload 173
- lccProgram constructors
 - Constructor 171
 - in lccProgram class 171
- lccProgramId
 - in lccProgramId class 175
 - in Resource identification classes 16
- lccProgramId class
 - Constructor 175
 - operator= 175
- lccProgramId constructors
 - Constructor 175
 - in lccProgramId class 175
- lccRBA
 - in Browsing records 30
 - in File control 27
 - in lccRBA class 177
 - in lccRecordIndex class 179
 - in Reading ESDS records 28
 - in Reading records 27
 - in Writing ESDS records 29

- lccRBA (*continued*)
 - in Writing records 28
 - in Writing RRDS records 29
- lccRBA class
 - Constructor 177
 - number 178
 - operator!= 178
 - operator= 177
 - operator== 177
 - reading records 27
- lccRBA constructor
 - Constructor 177
 - in lccRBA class 177
- lccRecordIndex
 - in C++ Exceptions and the Foundation Classes 51
 - in lccRecordIndex class 179
- lccRecordIndex class
 - Constructor 179
 - length 179
 - type 179
 - Type 180
- lccRecordIndex constructor (protected)
 - Constructor 179
 - in lccRecordIndex class 179
- lccRequestId
 - in Example of starting transactions 35, 36
 - in lccRequestId class 181
 - in Parameter passing conventions 61
- lccRequestId class
 - Constructor 181
 - operator= 181
- lccRequestId constructors
 - Constructor 181
 - in lccRequestId class 181
- lccResource
 - in Base classes 15
 - in Example of polymorphic behavior 59
 - in lccResource class 183
 - in Polymorphic Behavior 58
 - in Resource classes 17, 18
 - in Scope of data in lccBuf reference returned from 'read' methods 61
- lccResource class
 - actionOnCondition 183
 - ActionOnCondition 188
 - actionOnConditionAsChar 183
 - actionsOnConditionsText 184
 - clear 184
 - condition 184
 - conditionText 185
 - ConditionType 189
 - Constructor 183
 - get 185
 - handleEvent 185
 - HandleEventReturnOpt 188
 - id 185
 - isEDFOn 185
 - isRouteOptionOn 185
 - name 186
 - overview 15
 - put 186
- lccResource class (*continued*)
 - routeOption 186
 - setActionOnAnyCondition 186
 - setActionOnCondition 186
 - setActionsOnConditions 187
 - setEDF 187
 - setRouteOption 187, 188
 - working with subclasses 25
- lccResource constructor (protected)
 - Constructor 183
 - in lccResource class 183
- lccResourceId
 - in Base classes 15
 - in C++ Exceptions and the Foundation Classes 51
 - in Resource identification classes 16
- lccResourceId class
 - Constructor 191
 - name 191
 - nameLength 191
 - operator= 192
 - overview 15, 16
- lccResourceId constructors (protected)
 - Constructor 191
 - in lccResourceId class 191
- lccRRN
 - in Browsing records 30
 - in Deleting normal records 30
 - in File control 27
 - in lccRecordIndex class 179
 - in lccRRN class 193
 - in Reading records 27
 - in Reading RRDS records 28
 - in Writing records 28
- lccRRN class
 - Constructor 193
 - number 194
 - operator!= 194
 - operator= 193
 - operator== 193
 - reading records 27
- lccRRN constructors
 - Constructor 193
 - in lccRRN class 193
- lccSemaphore class
 - Constructor 195
 - lifeTime 195
 - LifeTime 197
 - lock 196
 - LockType 197
 - tryLock 196
 - type 196
 - unlock 196
- lccSemaphore constructor
 - Constructor 195
 - in lccSemaphore class 195
- lccSession
 - in Buffer objects 23
- lccSession class
 - allocate 200
 - AllocateOpt 208
 - connectProcess 200, 201

- lccSession class (*continued*)
 - Constructor 199, 200
 - converse 201
 - convId 201
 - errorCode 201
 - extractProcess 202
 - flush 202
 - free 202
 - get 202
 - isErrorSet 202
 - isNoDataSet 202
 - isSignalSet 203
 - issueAbend 203
 - issueConfirmation 203
 - issueError 203
 - issuePrepare 203
 - issueSignal 204
 - PIPList 204
 - process 204
 - put 204
 - receive 204
 - send 205
 - sendInvite 205
 - sendLast 206
 - SendOpt 208
 - state 206
 - StateOpt 208
 - stateText 207
 - syncLevel 207
 - SyncLevel 208
- lccSession constructor (protected)
 - Constructor 200
 - in lccSession class 199
- lccSession constructors (public)
 - Constructor 199
 - in lccSession class 199
- lccStartRequestQ
 - in Accessing start data 34
 - in Buffer objects 23
 - in Example of starting transactions 36, 37
 - in lccRequestId class 181
 - in lccStartRequestQ class 209
 - in Mapping EXEC CICS calls to Foundation Class methods 291
 - in Parameter passing conventions 61
 - in Singleton classes 20
 - in Starting transactions asynchronously 34
- lccStartRequestQ class
 - cancel 209
 - CheckOpt 214
 - clearData 209
 - Constructor 209
 - data 210
 - instance 210
 - overview 20
 - ProtectOpt 214
 - queueName 210
 - registerData 210
 - reset 210
 - retrieveData 211
 - RetrieveOpt 214
- lccStartRequestQ class (*continued*)
 - returnTermId 211
 - returnTransId 211
 - setData 211
 - setQueueName 211
 - setReturnTermId 212
 - setReturnTransId 212
 - setStartOpts 212
 - start 213
- lccStartRequestQ constructor (protected)
 - Constructor 209
 - in lccStartRequestQ class 209
- lccSysId
 - in lccSysId class 215
 - in Program control 33
- lccSysId class
 - Constructor 215
 - operator= 215
- lccSysId constructors
 - Constructor 215
 - in lccSysId class 215
- lccSystem
 - in Singleton classes 20
- lccSystem class
 - applName 217
 - beginBrowse 217, 218
 - Constructor 217
 - dateFormat 218
 - endBrowse 218
 - freeStorage 218
 - getFile 218, 219
 - getNextFile 219
 - getStorage 219
 - instance 220
 - operatingSystem 220
 - operatingSystemLevel 220
 - overview 20
 - release 220
 - releaseText 220
 - ResourceType 222
 - sysId 221
 - workArea 221
- lccSystem constructor (protected)
 - Constructor 217
 - in lccSystem class 217
- lccTask
 - in C++ Exceptions and the Foundation Classes 50
 - in Example of starting transactions 37
 - in lccAlarmRequestId class 83
 - in lccTask class 223
 - in Parameter level 55
 - in Singleton classes 20
 - in Support Classes 19
- lccTask class
 - abend 223
 - abendData 223
 - AbendDumpOpt 231
 - AbendHandlerOpt 231
 - commitUOW 224
 - Constructor 223
 - delay 224

- lccTask class *(continued)*
 - dump 224
 - DumpOpts 231
 - enterTrace 225
 - facilityType 225
 - FacilityType 232
 - freeStorage 225
 - getStorage 226
 - instance 226
 - isCommandSecurityOn 226
 - isCommitSupported 226
 - isResourceSecurityOn 227
 - isRestarted 227
 - isStartDataAvailable 227
 - number 227
 - overview 20
 - principalSysId 227
 - priority 228
 - rollBackUOW 228
 - setDumpOpts 228
 - setPriority 228
 - setWaitText 228
 - startType 229
 - StartType 232
 - StorageOpts 232
 - suspend 229
 - TraceOpt 233
 - transId 229
 - triggerDataQueueId 229
 - userId 229
 - waitExternal 230
 - waitOnAlarm 230
 - WaitPostType 233
 - WaitPurgeability 233
 - workArea 230
- lccTask Constructor (protected)
 - Constructor 223
 - in lccTask class 223
- lccTask::commitUOW
 - in Scope of data in lccBuf reference returned from 'read' methods 61
- lccTempstore
 - in Working with lccResource subclasses 25
- lccTempStore
 - in Automatic condition handling (callHandleEvent) 52, 53
 - in Buffer objects 23
 - in C++ Exceptions and the Foundation Classes 51
 - in Deleting items 40
 - in Example of polymorphic behavior 59
 - in Example of Temporary Storage 40, 41
 - in lccTempStore class 235
 - in Reading items 40
 - in Resource classes 17
 - in Temporary storage 39
 - in Transient Data 38
 - in Updating items 40
 - in Working with lccResource subclasses 25
 - in Writing items 40
- lccTempStore class
 - clear 236
- lccTempStore class *(continued)*
 - Constructor 235
 - empty 236
 - get 236
 - Location 239
 - NoSpaceOpt 239
 - numberOfItems 236
 - put 236
 - readItem 236
 - readNextItem 237
 - rewriteItem 237
 - writeItem 237, 238
- lccTempStore constructors
 - Constructor 235
 - in lccTempStore class 235
- lccTempStore::readItem
 - in Scope of data in lccBuf reference returned from 'read' methods 61
- lccTempStore::readNextItem
 - in Scope of data in lccBuf reference returned from 'read' methods 61
- lccTempStoreId
 - in Base classes 15
 - in Example of Temporary Storage 40, 41
 - in lccTempStoreId class 241
 - in Temporary storage 39
- lccTempStoreId class
 - Constructor 241
 - operator= 241
- lccTempStoreId constructors
 - Constructor 241
 - in lccTempStoreId class 241
- lccTermId
 - in Base classes 15
 - in C++ Exceptions and the Foundation Classes 51
 - in Example of starting transactions 35
 - in Example of terminal control 42
 - in lccTermId class 243
 - in Terminal control 42
- lccTermId class
 - Constructor 243
 - operator= 243
 - overview 15
- lccTermId constructors
 - Constructor 243
 - in lccTermId class 243
- lccTerminal
 - in Buffer objects 23
 - in Example of terminal control 42
 - in Finding out information about a terminal 42
 - in lccTerminalData class 259
 - in Receiving data from a terminal 42
 - in Resource classes 17
 - in Sending data to a terminal 42
 - in Singleton classes 20
 - in Terminal control 42
- lccTerminal class
 - AID 245
 - AIDVal 257
 - Case 258
 - clear 245

- lccTerminal class *(continued)*
 - Color 258
 - Constructor 245
 - cursor 245
 - data 246
 - erase 246
 - freeKeyboard 246
 - get 246
 - height 246
 - Highlight 258
 - inputCursor 246
 - instance 247
 - line 247
 - netName 247
 - NextTransIdOpt 258
 - operator« 247, 248, 249
 - put 249
 - receive 250
 - receive3270Data 250
 - registerInputMessage 173
 - send 250, 251
 - send3270Data 251, 252
 - sendLine 252, 253
 - setColor 253
 - setCursor 253, 254
 - setHighlight 254
 - setLine 254
 - setNewLine 254
 - setNextCommArea 255
 - setNextInputMessage 255
 - setNextTransId 255
 - signoff 255
 - signon 255, 256
 - waitForAID 256
 - width 257
 - workArea 257
- lccTerminal constructor (protected)
 - Constructor 245
 - in lccTerminal class 245
- lccTerminal::receive
 - in Scope of data in lccBuf reference returned from 'read' methods 61
- lccTerminalData
 - in Example of terminal control 42
 - in Finding out information about a terminal 42
 - in lccTerminalData class 259
 - in Terminal control 42
- lccTerminalData class
 - alternateHeight 259
 - alternateWidth 259
 - Constructor 259
 - defaultHeight 260
 - defaultWidth 260
 - graphicCharCodeSet 260
 - graphicCharSetId 260
 - isAPLKeyboard 260
 - isAPLText 261
 - isBTrans 261
 - isColor 261
 - isEWA 261
 - isExtended3270 261
- lccTerminalData class *(continued)*
 - isFieldOutline 262
 - isGoodMorning 262
 - isHighlight 262
 - isKatakana 262
 - isMSRControl 262
 - isPS 263
 - isSOSI 263
 - isTextKeyboard 263
 - isTextPrint 263
 - isValidation 263
- lccTerminalData constructor (protected)
 - Constructor 259
 - in lccTerminalData class 259
- lccTime
 - in Base classes 16
 - in lccTime class 265
 - in Parameter passing conventions 61
 - in Support Classes 19
- lccTime class
 - Constructor 265
 - hours 265
 - minutes 265
 - overview 16
 - seconds 265
 - timeInHours 266
 - timeInMinutes 266
 - timeInSeconds 266
 - type 266
 - Type 267
- lccTime constructor (protected)
 - Constructor 265
 - in lccTime class 265
- lccTimeInterval
 - in Base classes 16
 - in delay 224
 - in Example of starting transactions 35, 36
 - in lccTime class 265
 - in Support Classes 19
- lccTimeInterval class
 - Constructor 269
 - operator= 269
 - set 269
- lccTimeInterval constructors
 - Constructor 269
 - in lccTimeInterval class 269
- lccTimeOfDay
 - in Base classes 16
 - in delay 224
 - in lccTime class 265
 - in Support Classes 19
- lccTimeOfDay class
 - Constructor 271
 - operator= 271
 - set 271
- lccTimeOfDay constructors
 - Constructor 271
 - in lccTimeOfDay class 271
- lccTPNameId
 - in lccTPNameId class 273

- lccTPNameId class
 - Constructor 273
 - operator= 273
- lccTPNameId constructors
 - Constructor 273
 - in lccTPNameId class 273
- lccTransId
 - in Base classes 15
 - in Example of starting transactions 35
 - in lccResourceId class 191
 - in lccTransId class 275
 - in Parameter passing conventions 61
- lccTransId class
 - Constructor 275
 - operator= 275
 - overview 15
- lccTransId constructors
 - Constructor 275
 - in lccTransId class 275
- lccUser class
 - changePassword 277
 - Constructor 277
 - daysUntilPasswordExpires 278
 - ESMReason 278
 - ESMResponse 278
 - groupId 278
 - invalidPasswordAttempts 278
 - language 278
 - lastPasswordChange 278
 - lastUseTime 279
 - passwordExpiration 279
 - setLanguage 279
 - verifyPassword 279
- lccUser constructors
 - Constructor 277
 - in lccUser class 277
- lccUserControl
 - in C++ Exceptions and the Foundation Classes 50
 - in Example of file control 31
 - in Example of managing transient data 38
 - in Example of polymorphic behavior 58
 - in Example of starting transactions 35
 - in Example of Temporary Storage 41
 - in Example of terminal control 43
 - in Example of time and date services 44
 - in Hello World 9
 - in main function 287
 - in Program control 33
 - in Singleton classes 20
- lccUserControl class 9
- lccUserId
 - in lccUserId class 281
- lccUserId class
 - Constructor 281
 - operator= 281
- lccUserId constructors
 - Constructor 281
 - in lccUserId class 281
- lccValue structure
 - CVDA 283
- id
 - in lccResource class 185
- Id
 - in Resource identification classes 16
- id (parameter)
 - in Constructor 83, 121, 125, 133, 145, 151, 153, 157, 159, 165, 169, 171, 175, 181, 191, 195, 199, 215, 235, 241, 243, 273, 275, 277, 281
 - in getFile 219
 - in operator= 84, 119, 125, 145, 151, 157, 159, 165, 169, 175, 181, 192, 215, 241, 243, 273, 275, 281
 - in setJournalTypeId 154
 - in signon 255
 - in waitOnAlarm 230
- ifSOSReturnCondition
 - in StorageOpts 232
- ignoreAbendHandler
 - in AbendHandlerOpt 231
- immediate
 - in NextTransIdOpt 258
- index (parameter)
 - in Constructor 133, 147
 - in registerRecordIndex 139
 - in reset 148
- Inherited protected methods
 - in lccAbendData class 75
 - in lccAbsTime class 81
 - in lccAlarmRequestId class 85
 - in lccBuf class 99
 - in lccClock class 104
 - in lccConsole class 112
 - in lccControl class 117
 - in lccConvId class 120
 - in lccDataQueue class 123
 - in lccDataQueueId class 126
 - in lccEvent class 128
 - in lccException class 131
 - in lccFile class 142
 - in lccFileId class 146
 - in lccFileIterator class 149
 - in lccGroupId class 152
 - in lccJournal class 156
 - in lccJournalId class 158
 - in lccJournalTypeId class 160
 - in lccKey class 163
 - in lccLockId class 166
 - in lccMessage class 168
 - in lccPartnerId class 170
 - in lccProgram class 174
 - in lccProgramId class 176
 - in lccRBA class 178
 - in lccRecordIndex class 180
 - in lccRequestId class 182
 - in lccResource class 188
 - in lccResourceId class 192
 - in lccRRN class 194
 - in lccSemaphore class 197
 - in lccSession class 208
 - in lccStartRequestQ class 214
 - in lccSysId class 216
 - in lccSystem class 221

Inherited protected methods *(continued)*

- in lccTask class 231
 - in lccTempStore class 238
 - in lccTempStoreId class 242
 - in lccTermId class 244
 - in lccTerminal class 257
 - in lccTerminalData class 264
 - in lccTime class 266
 - in lccTimeInterval class 270
 - in lccTimeOfDay class 272
 - in lccTPNameId class 274
 - in lccTransId class 276
 - in lccUser class 280
 - in lccUserId class 282
- Inherited public methods
- in lccAbendData class 75
 - in lccAbsTime class 80
 - in lccAlarmRequestId class 84
 - in lccBuf class 99
 - in lccClock class 104
 - in lccConsole class 111
 - in lccControl class 116
 - in lccConvId class 119
 - in lccDataQueue class 123
 - in lccDataQueueId class 125
 - in lccEvent class 128
 - in lccException class 131
 - in lccFile class 142
 - in lccFileId class 145
 - in lccFileIterator class 148
 - in lccGroupId class 151
 - in lccJournal class 155
 - in lccJournalId class 158
 - in lccJournalTypeId class 159
 - in lccKey class 163
 - in lccLockId class 165
 - in lccMessage class 168
 - in lccPartnerId class 169
 - in lccProgram class 173
 - in lccProgramId class 175
 - in lccRBA class 178
 - in lccRecordIndex class 179
 - in lccRequestId class 182
 - in lccResource class 188
 - in lccResourceId class 192
 - in lccRRN class 194
 - in lccSemaphore class 196
 - in lccSession class 207
 - in lccStartRequestQ class 213
 - in lccSysId class 215
 - in lccSystem class 221
 - in lccTask class 231
 - in lccTempStore class 238
 - in lccTempStoreId class 241
 - in lccTermId class 243
 - in lccTerminal class 257
 - in lccTerminalData class 264
 - in lccTime class 266
 - in lccTimeInterval class 270
 - in lccTimeOfDay class 272
 - in lccTPNameId class 273

Inherited public methods *(continued)*

- in lccTransId class 275
 - in lccUser class 279
 - in lccUserId class 281
- initByte (parameter)
- in getStorage 219, 226
- initData
- in lccControl class 114
 - in Public methods 114
- initializeEnvironment
- in Functions 66
 - in lcc structure 66
 - in Method level 55
 - in Storage management 60
- initRBA (parameter)
- in Constructor 177
- initRRN (parameter)
- in Constructor 193
- initValue (parameter)
- in Constructor 161
- inputCursor
- in lccTerminal class 246
- insert
- in Example of Temporary Storage 41
 - in lccBuf class 94
 - in lccBuf constructors 24
- Installed contents
- Location 6
- instance
- in lccAbendData class 74
 - in lccConsole class 109
 - in lccControl class 114
 - in lccStartRequestQ class 210
 - in lccSystem class 220
 - in lccTask class 226
 - in lccTerminal class 247
 - in Singleton classes 20
- internal
- in DataAreaOwner 99
- internalError
- in C++ Exceptions and the Foundation Classes 51
 - in Type 132
- interval (parameter)
- in setReplyTimeout 110
- invalidArgument
- in C++ Exceptions and the Foundation Classes 50
 - in Type 131
- invalidMethodCall
- in C++ Exceptions and the Foundation Classes 51
 - in Type 131
- invalidPasswordAttempts
- in lccUser class 278
- IPMD 48
- isAddable
- in lccFile class 135
 - in Writing ESDS records 29
 - in Writing KSDS records 29
 - in Writing RRDS records 29
- isAPLKeyboard
- in lccTerminalData class 260
 - in Public methods 260

- isAPLText
 - in IccTerminalData class 261
 - in Public methods 261
- isBrowsable
 - in IccFile class 136
- isBTrans
 - in IccTerminalData class 261
- isClassMemoryMgmtOn
 - in Functions 66
 - in Icc structure 66
- isColor
 - in IccTerminalData class 261
- isCommandSecurityOn
 - in IccTask class 226
- isCommitSupported
 - in IccTask class 226
- isCreated
 - in IccControl class 114
- isDeletable
 - in IccFile class 136
- isDumpAvailable
 - in IccAbendData class 74
- isEDFOn
 - in Functions 66
 - in Icc structure 66
 - in IccResource class 185
- isEmptyOnOpen
 - in IccFile class 136
- isErrorSet
 - in IccSession class 202
- isEWA
 - in IccTerminalData class 261
- isExpired
 - in IccAlarmRequestId class 84
- isExtended3270
 - in IccTerminalData class 261
 - in Public methods 261
- isFamilySubsetEnforcementOn
 - in Functions 66
 - in Icc structure 66
- isFieldOutline
 - in IccTerminalData class 262
 - in Public methods 262
- isFMHContained
 - in IccBuf class 94
 - in Public methods 94
- isGoodMorning
 - in IccTerminalData class 262
 - in Public methods 262
- isHighlight
 - in IccTerminalData class 262
- isKatakana
 - in IccTerminalData class 262
- isMSRControl
 - in IccTerminalData class 262
- isNoDataSet
 - in IccSession class 202
- isPS
 - in IccTerminalData class 263
- ISR2
 - in Example of starting transactions 35
- isReadable
 - in IccFile class 136
 - in Reading ESDS records 28
 - in Reading KSDS records 28
 - in Reading RRDS records 28
- isReadable method 28
- isRecoverable
 - in IccFile class 137
- isResourceSecurityOn
 - in IccTask class 227
- isRestarted
 - in IccTask class 227
- isRouteOptionOn
 - in IccResource class 185
 - in Public methods 185
- isSignalSet
 - in IccSession class 203
- isSOSI
 - in IccTerminalData class 263
- isStartDataAvailable
 - in IccTask class 227
- issueAbend
 - in IccSession class 203
- issueConfirmation
 - in IccSession class 203
- issueError
 - in IccSession class 203
- issuePrepare
 - in IccSession class 203
- issueSignal
 - in IccSession class 204
- isTextKeyboard
 - in IccTerminalData class 263
 - in Public methods 263
- isTextPrint
 - in IccTerminalData class 263
 - in Public methods 263
- isUpdatable
 - in IccFile class 137
- isValidation
 - in IccTerminalData class 263
- item (parameter)
 - in rewriteItem 237
 - in writeItem 122, 237
- itemNum (parameter)
 - in readItem 236
 - in rewriteItem 237
- ITMP
 - in Example of starting transactions 35

J

- journalNum (parameter)
 - in Constructor 153, 157
 - in operator= 157
- journalTypeeld
 - in IccJournal class 154
- journalTypeName (parameter)
 - in Constructor 159
 - in operator= 159

jtypeid (parameter)
in setJournalTypeId 154

K

key
complete 28
generic 28
key (parameter)
in Constructor 161
in Example of file control 31
in operator!= 162
in operator= 162
in operator== 162
keyLength
in lccFile class 137
in Reading KSDS records 28
in Writing KSDS records 29
keyLength method 28
keyPosition
in lccFile class 137
in Reading KSDS records 28
in writing KSDS records 29
keyPosition method 28
kind
in lccKey class 162
Kind
in Enumerations 163
in lccKey class 163
kind (parameter)
in Constructor 161
in setKind 163
KSDS
in File control 27
KSDS file 27

L

language
in lccUser class 278
language (parameter)
in setLanguage 279
lastCommand
in StateOpt 208
lastPasswordChange
in lccUser class 278
lastUseTime
in lccUser class 279
length
in lccProgram class 172
in lccRecordIndex class 179
length (parameter)
in append 92
in assign 93, 161
in Constructor 91
in cut 93
in insert 94
in overlay 98
in replace 98
in setDataLength 98

level (parameter)
in connectProcess 200, 201
level0
in SyncLevel 208
level1
in SyncLevel 208
level2
in SyncLevel 208
life (parameter)
in Constructor 195
lifeTime
in lccSemaphore class 195
LifeTime
in Enumerations 197
in lccSemaphore class 197
line
in Finding out information about a terminal 42
in lccTerminal class 247
lineNum (parameter)
in setLine 254
link
in lccProgram class 172
load
in lccProgram class 173
LoadOpt
in Enumerations 174
in lccProgram class 174
loc (parameter)
in Constructor 235
Location
in Dynamic link library 6
in Enumerations 239
in Header files 6
in lccTempStore class 239
in Installed contents 6
in Sample source code 6
lock
in lccSemaphore class 196
LockType
in Enumerations 197
in lccSemaphore class 197

M

main
in C++ Exceptions and the Foundation Classes 49
in Example of file control 30
in Example of managing transient data 38
in Example of polymorphic behavior 58
in Example of starting transactions 35
in Example of Temporary Storage 40
in Example of terminal control 43
in Example of time and date services 44
in Header files 6
in main function 287
in Program control 33
in Storage management 60
main function
in Hello World 9
majorCode
in ConditionType 189

- manual
 - in UpdateMode 105
- Manual condition handling (noAction)
 - in CICS conditions 52
 - in Conditions, errors, and exceptions 52
- maxValue
 - in Range 107
- mem (parameter)
 - in initializeEnvironment 66
- memory
 - in Location 239
- message
 - in lccException class 130
- message (parameter)
 - in Constructor 129
 - in setNextInputMessage 255
- Method level
 - in Conditions, errors, and exceptions 55
 - in Platform differences 55
- methodName
 - in lccEvent class 128
 - in lccException class 130
 - in lccMessage class 167
- methodName (parameter)
 - in Constructor 127, 129, 167
- milliseconds
 - in lccAbsTime class 78
 - in lccClock class 103
- minorCode
 - in ConditionType 189
- minutes
 - in lccAbsTime class 78
 - in lccTime class 265
- minutes (parameter)
 - in Constructor 265, 269, 271
 - in set 269, 270, 271, 272
- Miscellaneous
 - Example of polymorphic behavior 58
- mixed
 - in Case 258
- mode (parameter)
 - in readNextRecord 147
 - in readPreviousRecord 148
 - in readRecord 138
- monthOfYear
 - in Example of time and date services 45
 - in lccAbsTime class 79
 - in lccClock class 103
- MonthOfYear
 - in Enumerations 105
 - in lccClock class 105
- msg (parameter)
 - in clearInputMessage 172
 - in registerInputMessage 173
 - in setInputMessage 173
- MVS/ESA
 - in ClassMemoryMgmt 68
 - in Storage management 60
- MVSPost
 - in WaitPostType 233

- MyTempStore
 - in Automatic condition handling (callHandleEvent) 53

N

- N
 - in operatingSystem 220
- name
 - in lccResource class 186
 - in lccResourceId class 191
- name (parameter)
 - in Constructor 83, 165, 215, 241, 243, 273, 275, 281
 - in operator= 165, 215, 241, 243, 273, 275, 281
 - in setWaitText 228
- nameLength
 - in lccResourceId class 191
- NameOpt
 - in Enumerations 89
 - in lccBase class 89
- netName
 - in lccTerminal class 247
- neutral
 - in Color 258
- new
 - in Storage management 60
- new operator 13
- newPassword (parameter)
 - in changePassword 277
 - in signon 255, 256
- NextTransIdOpt
 - in Enumerations 258
 - in lccTerminal class 258
- noAccess
 - in Access 143
- noAction
 - in ActionOnCondition 188
 - in CICS conditions 52
- noCommitOnReturn
 - in CommitOpt 174
- NONCICS
 - in ASRAKeyType 72
- none
 - in FacilityType 232
- noQueue
 - in AllocateOpt 208
- normal
 - in ReadMode 143
 - in SendOpt 208
 - in TraceOpt 233
- NoSpaceOpt
 - in Enumerations 239
 - in lccTempStore class 239
- noSuspend
 - in Options 156
- notAddable
 - in Access 142
- NOTAPPLIC
 - in ASRAKeyType 72
 - in ASRASpaceType 73

- NOTAPPLIC (*continued*)
 - in ASRAStorageType 73
- notBrowsable
 - in Access 142
- notDeletable
 - in Access 143
- notPurgeable
 - in WaitPurgeability 233
- notReadable
 - in Access 142
- notUpdatable
 - in Access 143
- num (parameter)
 - in operator!= 178
 - in operator« 97, 98, 248, 249
 - in operator= 177, 193
 - in operator== 177
- number
 - in lccException class 130
 - in lccJournalId class 157
 - in lccMessage class 167
 - in lccRBA class 178
 - in lccRRN class 194
 - in lccTask class 227
 - in Writing RRDS records 29
- number (parameter)
 - in Constructor 167
 - in setCustomClassNum 88
- numberOfItems
 - in lccTempStore class 236
- numEvents (parameter)
 - in waitExternal 230
- numLines (parameter)
 - in setNewLine 254
- numRoutes (parameter)
 - in setRouteCodes 110

O

- obj (parameter)
 - in Using an object 14
- object
 - creating 13
 - deleting 14
 - in GetOpt 68
 - using 14
- object (parameter)
 - in Constructor 127, 129
 - in operator delete 88
- Object level
 - in Conditions, errors, and exceptions 54
 - in Platform differences 54
- objectCreationError
 - in C++ Exceptions and the Foundation Classes 50
 - in Type 131
- offset (parameter)
 - in cut 93
 - in dataArea 93
 - in insert 94
 - in replace 98
 - in setCursor 253

- onOff (parameter)
 - in setEDF 67, 187
- open
 - in Status 143
- openStatus
 - in lccFile class 138
- operatingSystem
 - in lccSystem class 220
 - in Public methods 220
- operatingSystemLevel
 - in lccSystem class 220
- operator const char*
 - in lccBuf class 94
- operator delete
 - in lccBase class 88
 - in Public methods 88
- operator new
 - in lccBase class 88
- operator!=
 - in lccBuf class 96
 - in lccKey class 162, 163
 - in lccRBA class 178
 - in lccRRN class 194
 - in Public methods 96
- operator«
 - in lccBuf class 96, 98
 - in lccTerminal class 247, 248, 249
 - in Working with lccResource subclasses 25
- operator+=
 - in lccBuf class 95
- operator=
 - in Example of file control 31
 - in lccAbsTime class 79
 - in lccAlarmRequestId class 84
 - in lccBuf class 95
 - in lccConvId class 119
 - in lccDataQueueId class 125
 - in lccFileId class 145
 - in lccGroupId class 151
 - in lccJournalId class 157
 - in lccJournalTypeId class 159
 - in lccKey class 162
 - in lccLockId class 165
 - in lccPartnerId class 169
 - in lccProgramId class 175
 - in lccRBA class 177
 - in lccRequestId class 181
 - in lccResourceId class 192
 - in lccRRN class 193
 - in lccSysId class 215
 - in lccTempStoreId class 241
 - in lccTermId class 243
 - in lccTimeInterval class 269
 - in lccTimeOfDay class 271
 - in lccTPNameId class 273
 - in lccTransId class 275
 - in lccUserId class 281
 - in Protected methods 192
 - in Public methods 79, 269
 - in Working with lccResource subclasses 25

- operator==
 - in lccBuf class 95
 - in lccKey class 162
 - in lccRBA class 177
 - in lccRRN class 193
- opt (parameter)
 - in abendCode 71
 - in access 134
 - in accessMethod 134
 - in alternateHeight 259
 - in alternateWidth 259
 - in ASRAInterrupt 72
 - in ASRAKeyType 72
 - in ASRAPSW 72
 - in ASRARegisters 73
 - in ASRASpaceType 73
 - in ASRAStorageType 74
 - in className 87
 - in defaultHeight 260
 - in defaultWidth 260
 - in enableStatus 135
 - in enterTrace 225
 - in graphicCharCodeSet 260
 - in graphicCharSetId 260
 - in height 246
 - in isAddable 135
 - in isAPLKeyboard 260
 - in isAPLText 261
 - in isBrowsable 136
 - in isBTrans 261
 - in isColor 261
 - in isDeletable 136
 - in isDumpAvailable 74
 - in isEmptyOnOpen 136
 - in isEWA 261
 - in isExtended3270 261
 - in isFieldOutline 262
 - in isGoodMorning 262
 - in isHighlight 262
 - in isKatakana 262
 - in isMSRControl 262
 - in isPS 263
 - in isReadable 136
 - in isRecoverable 137
 - in isSOSI 263
 - in isTextKeyboard 263
 - in isTextPrint 263
 - in isUpdatable 137
 - in isValidation 263
 - in keyLength 137
 - in keyPosition 137
 - in link 172
 - in load 173
 - in openStatus 138
 - in originalAbendCode 74
 - in principalSysId 227
 - in priority 228
 - in programName 74
 - in recordFormat 139
 - in recordLength 139
 - in rewriteItem 237
- opt (parameter) (*continued*)
 - in setNextTransId 255
 - in type 141
 - in userId 229
 - in waitExternal 230
 - in width 257
 - in write 111
 - in writeAndGetReply 111
 - in writeItem 237, 238
- opt1 (parameter)
 - in abend 223
- opt2 (parameter)
 - in abend 223
- option (parameter)
 - in allocate 200
 - in retrieveData 211
 - in send 205
 - in sendInvite 205
 - in sendLast 206
 - in state 207
 - in stateText 207
 - in wait 155
 - in writeRecord 155
- Options
 - in Enumerations 156
 - in lccJournal class 156
- options (parameter)
 - in Constructor 153
- opts (parameter)
 - in setDumpOpts 228
- originalAbendCode
 - in lccAbendData class 74
- OS/2
 - in ClassMemoryMgmt 68
 - in Storage management 60
- OS/2, CICS
 - in Platform differences 54
- Other datasets for CICS/ESA
 - in Installed contents 7
- Output from sample programs
 - First Screen 305
 - Second Screen 305
- overlay
 - in lccBuf class 98
- overview of Foundation Classes 15
- Overview of the foundation classes
 - Calling methods on a resource object 21
 - Creating a resource object 19

P

P

- in operatingSystem 220
- PA1 to PA3
 - in AIDVal 258
- packedDecimal
 - in lccAbsTime class 79
- Parameter level
 - in Conditions, errors, and exceptions 55
 - in Platform differences 55
- parameter passing 60

- Parameter passing conventions
 - in Miscellaneous 60
- partnerName (parameter)
 - in Constructor 169
 - in operator= 169
- password (parameter)
 - in changePassword 277
 - in signon 255, 256
 - in verifyPassword 279
- passwordExpiration
 - in lccUser class 279
- PF1 to PF24
 - in AIDVal 258
- pink
 - in Color 258
- PIP (parameter)
 - in connectProcess 200, 201
- PIPList
 - in lccSession class 204
- platform differences
 - method level 55
 - object level 54
 - parameter level 55
- Platform differences
 - in Conditions, errors, and exceptions 54
 - Method level 55
 - Object level 54
 - Parameter level 55
- platformError
 - in Type 132
- Platforms
 - in Enumerations 69
 - in lcc structure 69
- polymorphic behavior 57
- Polymorphic Behavior
 - Example of polymorphic behavior 58
 - in Miscellaneous 57
- popt (parameter)
 - in setStartOpts 212
- prefix (parameter)
 - in registerPrefix 154
 - in setPrefix 154
- pri (parameter)
 - in setPriority 228
- principalSysId
 - in lccTask class 227
 - in Public methods 227
- print
 - in Polymorphic Behavior 58
- priority
 - in lccTask class 228
 - in Public methods 228
- process
 - in lccSession class 204
- profile (parameter)
 - in Constructor 199
- progName (parameter)
 - in Constructor 171, 175
 - in operator= 175
- program control
 - example 32
- program control (*continued*)
 - introduction 32
- Program control
 - in Using CICS Services 32
- programId
 - in lccControl class 114
 - in Method level 55
 - in Public methods 114
- programId (parameter)
 - in setAbendHandler 115
- programName
 - in lccAbendData class 74
 - in Public methods 74
- programName (parameter)
 - in setAbendHandler 115
- Protected methods
 - in lccBase class 88
 - in lccResourceId class 192
 - operator= 192
 - setClassName 88
 - setCustomClassNum 88
- ProtectOpt
 - in Enumerations 214
 - in lccStartRequestQ class 214
- pStorage (parameter)
 - in freeStorage 218
- Public methods
 - abend 223
 - abendCode 71
 - abendData 223
 - absTime 101
 - access 134
 - accessMethod 134
 - actionOnCondition 183
 - actionOnConditionAsChar 183
 - actionsOnConditionsText 184
 - address 171
 - AID 245
 - allocate 200
 - alternateHeight 259
 - alternateWidth 259
 - append 92
 - applName 217
 - ASRAInterrupt 71
 - ASRAKeyType 72
 - ASRAPSW 72
 - ASRARegisters 72
 - ASRASpaceType 73
 - ASRAStorageType 74
 - assign 92, 93, 161
 - beginBrowse 217, 218
 - beginInsert(VSAM only) 134
 - callingProgramId 113
 - cancel 209
 - cancelAbendHandler 113
 - cancelAlarm 101
 - changePassword 277
 - className 87, 127, 130, 167
 - classType 87, 127, 130
 - clear 121, 184, 236, 245
 - clearData 209

Public methods (*continued*)

clearInputMessage 171
clearPrefix 154
commArea 114
commitUOW 224
completeLength 161
condition 127, 184
conditionText 128, 185
connectProcess 200, 201
console 114
converse 201
convld 201
cursor 245
customClassNum 88
cut 93
data 210, 246
dataArea 93
dataAreaLength 93
dataAreaOwner 93
dataAreaType 94
dataLength 94
date 77, 102
dateFormat 218
dayOfMonth 78, 102
dayOfWeek 78, 102
daysSince1900 78, 102
daysUntilPasswordExpires 278
defaultHeight 260
defaultWidth 260
delay 224
deleteLockedRecord 134
deleteRecord 135
dump 224
empty 121, 236
enableStatus 135
endBrowse 218
endInsert(VSAM only) 135
enterTrace 225
entryPoint 172
erase 246
errorCode 201
ESMReason 278
ESMResponse 278
extractProcess 202
facilityType 225
flush 202
free 202
freeKeyboard 246
freeStorage 218, 225
get 122, 185, 202, 236, 246
getFile 218, 219
getNextFile 219
getStorage 219, 226
graphicCharCodeSet 260
graphicCharSetId 260
groupid 278
handleEvent 185
height 246
hours 78, 265
id 185
in lccAbendData class 71

Public methods (*continued*)

in lccAbsTime class 77
in lccAlarmRequestId class 83
in lccBase class 87
in lccBuf class 92
in lccClock class 101
in lccConsole class 109
in lccControl class 113
in lccConvld class 119
in lccDataQueue class 121
in lccDataQueueId class 125
in lccEvent class 127
in lccException class 130
in lccFile class 133
in lccFileId class 145
in lccFileIterator class 147
in lccGroupId class 151
in lccJournal class 153
in lccJournalId class 157
in lccJournalTypeId class 159
in lccKey class 161
in lccLockId class 165
in lccMessage class 167
in lccPartnerId class 169
in lccProgram class 171
in lccProgramId class 175
in lccRBA class 177
in lccRecordIndex class 179
in lccRequestId class 181
in lccResource class 183
in lccResourceId class 191
in lccRRN class 193
in lccSemaphore class 195
in lccSession class 200
in lccStartRequestQ class 209
in lccSysId class 215
in lccSystem class 217
in lccTask class 223
in lccTempStore class 235
in lccTempStoreId class 241
in lccTermId class 243
in lccTerminal class 245
in lccTerminalData class 259
in lccTime class 265
in lccTimeInterval class 269
in lccTimeOfDay class 271
in lccTPNameId class 273
in lccTransId class 275
in lccUser class 277
in lccUserId class 281
initData 114
inputCursor 246
insert 94
instance 74, 109, 114, 210, 220, 226, 247
invalidPasswordAttempts 278
isAddable 135
isAPLKeyboard 260
isAPLText 261
isBrowsable 136
isBTrans 261
isColor 261

Public methods *(continued)*

isCommandSecurityOn 226
isCommitSupported 226
isCreated 114
isDeletable 136
isDumpAvailable 74
isEDFOn 185
isEmptyOnOpen 136
isErrorSet 202
isEWA 261
isExpired 84
isExtended3270 261
isFieldOutline 262
isFMHContained 94
isGoodMorning 262
isHighlight 262
isKatakana 262
isMSRControl 262
isNoDataSet 202
isPS 263
isReadable 136
isRecoverable 137
isResourceSecurityOn 227
isRestarted 227
isRouteOptionOn 185
isSignalSet 203
isSOSI 263
isStartDataAvailable 227
issueAbend 203
issueConfirmation 203
issueError 203
issuePrepare 203
issueSignal 204
isTextKeyboard 263
isTextPrint 263
isUpdatable 137
isValidation 263
journalTypeid 154
keyLength 137
keyPosition 137
kind 162
language 278
lastPasswordChange 278
lastUseTime 279
length 172, 179
lifeTime 195
line 247
link 172
load 173
lock 196
message 130
methodName 128, 130, 167
milliSeconds 78, 103
minutes 78, 265
monthOfYear 79, 103
name 186, 191
nameLength 191
netName 247
number 130, 157, 167, 178, 194, 227
numberOfItems 236
openStatus 138

Public methods *(continued)*

operatingSystem 220
operatingSystemLevel 220
operator const char* 94
operator delete 88
operator new 88
operator!= 96, 162, 163, 178, 194
operator« 96, 98, 247, 248, 249
operator+= 95
operator= 79, 84, 95, 119, 125, 145, 151, 157, 159,
162, 165, 169, 175, 177, 181, 193, 215, 241, 243,
269, 271, 273, 275, 281
operator== 95, 162, 177, 193
originalAbendCode 74
overlay 98
packedDecimal 79
passwordExpiration 279
PIPList 204
principalSysId 227
priority 228
process 204
programId 114
programName 74
put 109, 122, 154, 186, 204, 236, 249
queueName 210
readItem 122, 236
readNextItem 237
readNextRecord 147
readPreviousRecord 148
readRecord 138
receive 204, 250
receive3270Data 250
recordFormat 138
recordIndex 139
recordLength 139
registerData 210
registerInputMessage 173
registerPrefix 154
registerRecordIndex 139
release 220
releaseText 220
replace 98
replyTimeout 109
reset 148, 210
resetAbendHandler 115
resetRouteCodes 110
retrieveData 211
returnProgramId 115
returnTermId 211
returnTransId 211
rewriteItem 237
rewriteRecord 139
rollBackUOW 228
routeOption 186
run 115
seconds 79, 265
send 205, 250, 251
send3270Data 251, 252
sendInvite 205
sendLast 206
sendLine 252, 253

Public methods (*continued*)

- session 115
- set 269, 271
- setAbendHandler 115
- setAccess 140
- setActionOnAnyCondition 186
- setActionOnCondition 186
- setActionsOnConditions 187
- setAlarm 103
- setAllRouteCodes 110
- setColor 253
- setCursor 253, 254
- setData 211
- setDataLength 98
- setDumpOpts 228
- setEDF 187
- setEmptyOnOpen 140
- setFMHContained 99
- setHighlight 254
- setInputMessage 173
- setJournalTypeId 154
- setKind 163
- setLanguage 279
- setLine 254
- setNewLine 254
- setNextCommArea 255
- setNextInputMessage 255
- setNextTransId 255
- setPrefix 154
- setPriority 228
- setQueueName 211
- setReplyTimeout 110
- setReturnTermId 212
- setReturnTransId 212
- setRouteCodes 110
- setRouteOption 187, 188
- setStartOpts 212
- setStatus 140
- setTimerECA 84
- setWaitText 228
- signoff 255
- signon 255, 256
- start 213
- startRequestQ 116
- startType 229
- state 206
- stateText 207
- summary 128, 130, 168
- suspend 229
- syncLevel 207
- sysId 221
- system 116
- task 116
- terminal 116
- text 168
- time 79, 103
- timeInHours 79, 266
- timeInMinutes 80, 266
- timeInSeconds 80, 266
- timerECA 84
- transId 229

Public methods (*continued*)

- triggerDataQueueId 229
- tryLock 196
- type 131, 141, 179, 196, 266
- typeText 131
- unload 173
- unlock 196
- unlockRecord 141
- update 104
- userId 229
- value 163
- verifyPassword 279
- wait 155
- waitExternal 230
- waitForAID 256
- waitOnAlarm 230
- width 257
- workArea 221, 230, 257
- write 110
- writeAndGetReply 111
- writeln 122, 237, 238
- writeRecord 141, 155
- year 80, 104
- purgeable
 - in WaitPurgeability 233
- put
 - in Example of polymorphic behavior 59
 - in lccConsole class 109
 - in lccDataQueue class 122
 - in lccJournal class 154
 - in lccResource class 186
 - in lccSession class 204
 - in lccTempStore class 236
 - in lccTerminal class 249
 - in Polymorphic Behavior 58

Q

- queue
 - in AllocateOpt 208
 - in NextTransIdOpt 258
- queueName
 - in Accessing start data 34
 - in lccStartRequestQ class 210
- queueName (parameter)
 - in Constructor 121, 125
 - in operator= 125
 - in setQueueName 212

R

- rAbendTask
 - in HandleEventReturnOpt 189
- Range
 - in Enumerations 107
 - in lccCondition structure 107
- RBA 27
- rba (parameter)
 - in operator!= 178
 - in operator= 177
 - in operator== 177

- rContinue
 - in HandleEventReturnOpt 188
- readable
 - in Access 142
- reading data 38
- Reading data
 - in Transient Data 38
 - in Using CICS Services 38
- Reading ESDS records
 - in File control 28
 - in Reading records 28
- reading items 40
- Reading items
 - in Temporary storage 40
 - in Using CICS Services 40
- Reading KSDS records
 - in File control 28
 - in Reading records 28
- Reading records
 - in File control 27
 - in Using CICS Services 27
- Reading ESDS records 28
- Reading KSDS records 28
- Reading RRDS records 28
- Reading RRDS records
 - in File control 28
 - in Reading records 28
- readItem
 - in Example of Temporary Storage 41
 - in lccDataQueue class 122
 - in lccTempStore class 236
 - in Reading data 38
 - in Reading items 40
 - in Scope of data in lccBuf reference returned from 'read' methods 61
 - in Temporary storage 39
 - in Transient Data 37
 - in Working with lccResource subclasses 25
- ReadMode
 - in Enumerations 143
 - in lccFile class 143
- readNextItem
 - in lccTempStore class 237
 - in Scope of data in lccBuf reference returned from 'read' methods 61
 - in Temporary storage 39
- readNextRecord
 - in Browsing records 30
 - in lccFileIterator class 147
 - in Public methods 147
- readNextRecord method 30
- READONLY
 - in ASRAStorageType 73
- readPreviousRecord 30
 - in Browsing records 30
 - in lccFileIterator class 148
- readRecord
 - in C++ Exceptions and the Foundation Classes 51
 - in Deleting locked records 30
 - in lccFile class 138
 - in Reading records 28
- readRecord (*continued*)
 - in Updating records 29
- readRecord method 28
- receive
 - in lccSession class 204
 - in lccTerminal class 250
 - in Receiving data from a terminal 42
- receive3270data
 - in Receiving data from a terminal 42
- receive3270Data
 - in lccTerminal class 250
 - in Public methods 250
- receiving data from a terminal 42
- Receiving data from a terminal
 - in Terminal control 42
 - in Using CICS Services 42
- record (parameter)
 - in writeRecord 155
- recordFormat
 - in lccFile class 138
 - in Reading ESDS records 28
 - in Reading RRDS records 28
 - in Writing ESDS records 29
 - in Writing RRDS records 29
- recordFormat method 28
- recordIndex
 - in lccFile class 139
 - in Reading ESDS records 28
 - in Reading KSDS records 28
 - in Reading RRDS records 28
 - in Writing ESDS records 29
 - in Writing KSDS records 29
 - in Writing RRDS records 29
- recordIndex method 28
- recordLength
 - in lccFile class 139
 - in Reading ESDS records 28
 - in Reading KSDS records 28
 - in Reading RRDS records 28
 - in Writing ESDS records 29
 - in Writing KSDS records 29
 - in Writing RRDS records 29
- recordLength method 28
- red
 - in Color 258
- registerData 210
 - in Example of starting transactions 36
 - in lccStartRequestQ class 210
 - in Starting transactions 34
- registerInputMessage 171
 - in lccTerminal class 173
- registerPrefix
 - in lccJournal class 154
 - in Public methods 154
- registerRecordIndex 28
 - in lccFile class 139
 - in Reading ESDS records 28
 - in Reading KSDS records 28
 - in Reading RRDS records 28
 - in Writing ESDS records 29
 - in Writing KSDS records 29

- registerRecordIndex (*continued*)
 - in Writing records 28
 - in Writing RRDS records 29
- registerRecordIndex method 28
- relative byte address 27
- relative record number 27
- release
 - in lccSystem class 220
- releaseAtTaskEnd
 - in LoadOpt 174
- releaseText
 - in lccSystem class 220
- remoteTermId
 - in Example of starting transactions 35
- replace
 - in lccBuf class 98
 - in lccBuf constructors 24
- replyTimeout
 - in lccConsole class 109
- req
 - in Example of starting transactions 36
- req1
 - in Example of starting transactions 35
- req2
 - in Example of starting transactions 35
- requestName (parameter)
 - in operator= 182
- reqId (parameter)
 - in cancel 209
 - in cancelAlarm 101
 - in delay 224
 - in setAlarm 103
 - in start 213
- requestName (parameter)
 - in Constructor 181
 - in operator= 84, 182
- requestNum (parameter)
 - in wait 155
- reset
 - in Browsing records 30
 - in lccFileIterator class 148
 - in lccStartRequestQ class 210
- resetAbendHandler
 - in lccControl class 115
- resetRouteCodes
 - in lccConsole class 110
 - in Public methods 110
- resId (parameter)
 - in beginBrowse 217
- resName (parameter)
 - in beginBrowse 218
 - in Constructor 191
- resource (parameter)
 - in beginBrowse 217, 218
 - in Constructor 195
 - in endBrowse 218
 - in enterTrace 225
- resource class 17
- Resource classes
 - in Overview of the foundation classes 17
- resource identification class 16
- Resource identification classes
 - in Overview of the foundation classes 16
- resource object
 - creating 19
- ResourceType
 - in Enumerations 222
 - in lccSystem class 222
- respectAbendHandler
 - in AbendHandlerOpt 231
- retrieveData
 - in Accessing start data 34
 - in lccStartRequestQ class 209, 211
 - in Mapping EXEC CICS calls to Foundation Class methods 291
- RetrieveOpt
 - in Enumerations 214
 - in lccStartRequestQ class 214
- return
 - in Mapping EXEC CICS calls to Foundation Class methods 291
- returnCondition
 - in NoSpaceOpt 239
- returnProgramId
 - in lccControl class 115
 - in Public methods 115
- returnTermId
 - in Accessing start data 34
 - in lccStartRequestQ class 211
- returnToCICS
 - in Functions 66
 - in lcc structure 66
- returnTransId
 - in Accessing start data 34
 - in lccStartRequestQ class 211
- reverse
 - in Highlight 258
- rewriteltem
 - in Example of Temporary Storage 41
 - in lccTempStore class 237
 - in Temporary storage 39
 - in Updating items 40
 - in Writing items 40
- rewriteRecord
 - in lccFile class 139
 - in Updating records 29
- rewriteRecord method 29
- rewriting records 29
- rollBackUOW
 - in lccTask class 228
- routeOption
 - in lccResource class 186
- row (parameter)
 - in send 251
 - in setCursor 254
- RRDS file
 - in File control 27
- RRN 27
- rrn (parameter)
 - in operator!= 194
 - in operator= 193
 - in operator== 193

- rThrowException
 - in HandleEventReturnOpt 188
- run
 - in Base classes 15
 - in C++ Exceptions and the Foundation Classes 50
 - in Example of file control 31, 32
 - in Example of managing transient data 38, 39
 - in Example of polymorphic behavior 58
 - in Example of starting transactions 35
 - in Example of Temporary Storage 41
 - in Example of terminal control 43
 - in Example of time and date services 44, 45
 - in Hello World 10
 - in lccControl class 113, 115
 - in main function 287, 288
 - in Mapping EXEC CICS calls to Foundation Class methods 291
 - in Program control 33
- 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 61
- Scope of data in lccBuf reference returned from 'read' methods
 - in Miscellaneous 61
- scope of references 61
- search (parameter)
 - in Constructor 147
 - in reset 148
- SearchCriterion
 - in Enumerations 143
 - in lccFile class 143
- Second Screen
 - in ICC\$PRG1 (IPR1) 305
 - in Output from sample programs 305
- seconds
 - in lccAbsTime class 79
 - in lccTime class 265
- seconds (parameter)
 - in Constructor 265, 269, 271
 - in set 269, 270, 271, 272
 - in setReplyTimeout 110
- send
 - in Example of terminal control 43
 - in Hello World 10
 - in lccSession class 205
 - in lccTerminal class 250, 251
 - in Sending data to a terminal 42
- send (parameter)
 - in converse 201
 - in put 109

- send (parameter) (*continued*)
 - in send 205
 - in sendInvite 205
 - in sendLast 206
 - in write 110, 111
 - in writeAndGetReply 111
- send3270Data
 - in lccTerminal class 251, 252
- sending data to a terminal 42
- Sending data to a terminal
 - in Terminal control 42
 - in Using CICS Services 42
- sendInvite
 - in lccSession class 205
- sendLast
 - in lccSession class 206
- sendLine
 - in Example of file control 31
 - in Example of terminal control 43
 - in lccTerminal class 252, 253
 - in Sending data to a terminal 42
- SendOpt
 - in Enumerations 208
 - in lccSession class 208
- sequential reading of files 30
- session
 - in FacilityType 232
 - in lccControl class 115
- set
 - in lccTimeInterval class 269
 - in lccTimeOfDay class 271
- set (parameter)
 - in boolText 65
- set...
 - in Sending data to a terminal 42
- setAbendHandler
 - in lccControl class 115
- setAccess
 - in lccFile class 140
- setActionOnAnyCondition
 - in lccResource class 186
- setActionOnCondition
 - in lccResource class 186
- setActionsOnConditions
 - in lccResource class 187
- setAlarm
 - in lccAlarmRequestId class 83
 - in lccClock class 103
- setAllRouteCodes
 - in lccConsole class 110
- setClassName
 - in lccBase class 88
 - in Protected methods 88
- setColor
 - in Example of terminal control 43
 - in lccTerminal class 253
- setCursor
 - in lccTerminal class 253, 254
- setCustomClassNum
 - in lccBase class 88
 - in Protected methods 88

- setData 210
 - in lccStartRequestQ class 211
 - in Starting transactions 34
- setDataLength
 - in lccBuf class 98
- setDumpOpts
 - in lccTask class 228
- setEDF
 - in Functions 67
 - in lcc structure 67
 - in lccResource class 187
- setEmptyOnOpen
 - in lccFile class 140
 - in Public methods 140
- setFMHContained
 - in lccBuf class 99
 - in Public methods 99
- setHighlight
 - in Example of terminal control 43
 - in lccTerminal class 254
- setInputMessage 171
 - in lccProgram class 173
 - in Public methods 173
- setJournalTypeld
 - in lccJournal class 154
- setKind
 - in Example of file control 31
 - in lccKey class 163
- setLanguage
 - in lccUser class 279
- setLine
 - in lccTerminal class 254
- setNewLine
 - in lccTerminal class 254
- setNextCommArea
 - in lccTerminal class 255
 - in Public methods 255
- setNextInputMessage
 - in lccTerminal class 255
- setNextTransld
 - in lccTerminal class 255
- setPrefix
 - in lccJournal class 154
- setPriority
 - in lccTask class 228
 - in Public methods 228
- setQueueName
 - in Example of starting transactions 36
 - in lccStartRequestQ class 211
 - in Starting transactions 34
- setReplyTimeout
 - in lccConsole class 110
- setReturnTermld
 - in Example of starting transactions 36
 - in lccStartRequestQ class 212
 - in Starting transactions 34
- setReturnTransld
 - in Example of starting transactions 36
 - in lccStartRequestQ class 212
 - in Starting transactions 34
- setRouteCodes
 - in lccConsole class 110
- setRouteOption
 - in Example of starting transactions 36, 37
 - in lccResource class 187, 188
 - in Program control 33
 - in Public methods 187, 188
- setStartOpts
 - in lccStartRequestQ class 212
- setStatus
 - in lccFile class 140
- setTimerECA
 - in lccAlarmRequestld class 84
- setWaitText
 - in lccTask class 228
- Severe error handling (abendTask)
 - in CICS conditions 54
 - in Conditions, errors, and exceptions 54
- SeverityOpt
 - in Enumerations 112
 - in lccConsole class 112
- signoff
 - in lccTerminal class 255
- signon
 - in lccTerminal class 255, 256
 - in Public methods 255, 256
- singleton class 20
- Singleton classes
 - in Creating a resource object 20
 - in Using CICS resources 20
- size (parameter)
 - in getStorage 219, 226
 - in operator new 88
- start
 - in Example of starting transactions 37
 - in lccRequestld class 181
 - in lccStartRequestQ class 209, 213
 - in Mapping EXEC CICS calls to Foundation Class methods 291
 - in Parameter passing conventions 61
 - in Starting transactions 34
- Starting transactions
 - in Starting transactions asynchronously 34
 - in Using CICS Services 34
- starting transactions asynchronously 34
- Starting transactions asynchronously
 - Accessing start data 34
 - Cancelling unexpired start requests 34
 - Example of starting transactions 35
 - in Using CICS Services 34
 - Starting transactions 34
- startIO
 - in Options 156
- startRequest
 - in StartType 232
- startRequestQ
 - in Example of starting transactions 36
 - in lccControl class 116
- startType
 - in Example of starting transactions 37
 - in lccTask class 229

- StartType
 - in Enumerations 232
 - in lccTask class 232
- state
 - in lccSession class 206
- StateOpt
 - in Enumerations 208
 - in lccSession class 208
- stateText
 - in lccSession class 207
- Status
 - in Enumerations 143
 - in lccFile class 143
- status (parameter)
 - in setStatus 140
- Storage management
 - in Miscellaneous 59
- StorageOpts
 - in Enumerations 232
 - in lccTask class 232
- storageOpts (parameter)
 - in getStorage 219, 226
- storeName (parameter)
 - in Constructor 235
- SUBSPACE
 - in ASRASpaceType 73
- summary
 - in lccEvent class 128
 - in lccException class 130
 - in lccMessage class 168
- support classes 18
- Support Classes
 - in Overview of the foundation classes 18
- suppressDump
 - in AbendDumpOpt 231
- suspend
 - in lccTask class 229
 - in NoSpaceOpt 239
- symbolic debuggers 48
- Symbolic Debuggers
 - in Compiling, executing, and debugging 48
 - in Debugging Programs 48
- synchronous
 - in Options 156
- syncLevel
 - in lccSession class 207
- SyncLevel
 - in Enumerations 208
 - in lccSession class 208
- sysId
 - in lccSystem class 221
- sysId (parameter)
 - in Constructor 199
 - in setRouteOption 187
- sysName (parameter)
 - in Constructor 199
 - in setRouteOption 188
- system
 - in lccControl class 116

T

- task
 - in lccControl class 116
 - in LifeTime 197
- temporary storage
 - deleting items 40
 - example 40
 - introduction 39
 - reading items 40
 - updating items 40
 - Writing items 40
- Temporary storage
 - Deleting items 40
 - Example of Temporary Storage 40
 - in Using CICS Services 39
 - Reading items 40
 - Updating items 40
 - Writing items 40
- termId (parameter)
 - in setReturnTermId 212
 - in start 213
- terminal
 - finding out about 42
 - in FacilityType 232
 - in Hello World 9
 - in lccControl class 116
 - receiving data from 42
 - sending data to 42
- terminal control
 - example 42
 - finding out information 42
 - introduction 42
 - receiving data 42
 - sending data 42
- Terminal control
 - Example of terminal control 42
 - Finding out information about a terminal 42
 - in Using CICS Services 42
 - Receiving data from a terminal 42
 - Sending data to a terminal 42
- terminalInput
 - in StartType 232
- termName (parameter)
 - in setReturnTermId 212
- Test
 - in C++ Exceptions and the Foundation Classes 49, 50
- test (parameter)
 - in boolText 65
- text
 - in lccMessage class 168
- text (parameter)
 - in Constructor 91, 92, 167
 - in operator!= 163
 - in operator« 96, 97, 248
 - in operator+= 95
 - in operator= 95
 - in operator== 162
 - in writeltem 122, 238
- throw
 - in C++ Exceptions and the Foundation Classes 49

- throw (*continued*)
 - in Exception handling (throwException) 53
- throwException
 - in ActionOnCondition 188
 - in CICS conditions 52
- ti
 - in Example of starting transactions 35, 36
- time
 - in lccAbsTime class 79
 - in lccClock class 103
- time (parameter)
 - in Constructor 77, 269, 271
 - in delay 224
 - in setAlarm 103
 - in start 213
- Time and date services
 - Example of time and date services 44
 - in Using CICS Services 44
- time services 44
- timeInHours
 - in lccAbsTime class 79
 - in lccTime class 266
- timeInMinutes
 - in lccAbsTime class 80
 - in lccTime class 266
- timeInSeconds
 - in lccAbsTime class 80
 - in lccTime class 266
- timeInterval
 - in Type 267
- timeInterval (parameter)
 - in operator= 269
- timeOfDay
 - in Type 267
- timeOfDay (parameter)
 - in operator= 271
- timerECA
 - in lccAlarmRequestId class 84
- timerECA (parameter)
 - in Constructor 83
 - in setTimerECA 84
- timeSeparator (parameter)
 - in time 79, 103
- TPName (parameter)
 - in connectProcess 201
- traceNum (parameter)
 - in enterTrace 225
- TraceOpt
 - in Enumerations 233
 - in lccTask class 233
- tracing
 - activating trace output 48
- Tracing a Foundation Class Program
 - Activating the trace output 48
 - in Compiling, executing, and debugging 48
 - in Debugging Programs 48
- transId
 - in lccTask class 229
- transId (parameter)
 - in setNextTransId 255
- transId (parameter)
 - in cancel 209
 - in connectProcess 200
 - in link 172
 - in setNextTransId 255
 - in setReturnTransId 212
 - in start 213
- transient data
 - deleting queues 38
 - example 38
 - introduction 37
 - reading data 38
 - Writing data 38
- Transient Data
 - Deleting queues 38
 - Example of managing transient data 38
 - in Using CICS Services 37
 - Reading data 38
 - Writing data 38
- transName (parameter)
 - in setReturnTransId 212
- triggerDataQueueId
 - in lccTask class 229
- trueFalse (parameter)
 - in setEmptyOnOpen 140
- try
 - in C++ Exceptions and the Foundation Classes 49, 50
 - in Exception handling (throwException) 53
 - in main function 288
- tryLock
 - in lccSemaphore class 196
- tryNumber
 - in C++ Exceptions and the Foundation Classes 49, 50
- type
 - in C++ Exceptions and the Foundation Classes 50
 - in lccException class 131
 - in lccFile class 141
 - in lccRecordIndex class 179
 - in lccSemaphore class 196
 - in lccTime class 266
- Type
 - in Enumerations 131, 180, 267
 - in lccException class 131
 - in lccRecordIndex class 180
 - in lccTime class 267
- type (parameter)
 - in condition 128, 184
 - in Constructor 87, 91, 92, 179, 191, 195
 - in waitExternal 230
- typeText
 - in lccException class 131

U

- underscore
 - in Highlight 258
- UNIX
 - in ClassMemoryMgmt 68
 - in Storage management 60

- unknownException
 - in Functions 67
 - in lcc structure 67
- unload
 - in lccProgram class 173
- unlock
 - in lccSemaphore class 196
- unlockRecord
 - in lccFile class 141
- UOW
 - in LifeTime 197
- updatable
 - in Access 142
- update
 - in lccClock class 104
 - in ReadMode 143
- update (parameter)
 - in Constructor 101
- UpdateMode
 - in Enumerations 105
 - in lccClock class 105
- updateToken (parameter)
 - in deleteLockedRecord 134
 - in readNextRecord 147, 148
 - in readPreviousRecord 148
 - in readRecord 138
 - in rewriteRecord 139, 140
 - in unlockRecord 141
- updating items 40
- Updating items
 - in Temporary storage 40
 - in Using CICS Services 40
- updating records 29
- Updating records
 - in File control 29
 - in Using CICS Services 29
- upper
 - in Case 258
- USER
 - in ASRAStorageType 73
- user (parameter)
 - in signon 256
- userDataKey
 - in StorageOpts 233
- USEREXECKEY
 - in ASRAKeyType 72
- userId
 - in lccTask class 229
- userId (parameter)
 - in start 213
- userName (parameter)
 - in Constructor 277
- Using an object
 - in C++ Objects 14
- using CICS resources 19
- Using CICS resources
 - Calling methods on a resource object 21
 - Creating a resource object 19
 - in Overview of the foundation classes 19
 - Singleton classes 20

- Using CICS Services
 - Accessing start data 34
 - Browsing records 30
 - Cancelling unexpired start requests 34
 - Deleting items 40
 - Deleting queues 38
 - Deleting records 30
 - Example of file control 30
 - Example of managing transient data 38
 - Example of starting transactions 35
 - Example of Temporary Storage 40
 - Example of terminal control 42
 - Example of time and date services 44
 - Finding out information about a terminal 42
 - Reading data 38
 - Reading items 40
 - Reading records 27
 - Receiving data from a terminal 42
 - Sending data to a terminal 42
 - Starting transactions 34
 - Updating items 40
 - Updating records 29
 - Writing data 38
 - Writing items 40
 - Writing records 28

V

- value
 - in lccKey class 163
- value (parameter)
 - in operator= 162
- verifyPassword
 - in lccUser class 279
 - in Public methods 279
- VSAM 27

W

- wait
 - in lccJournal class 155
 - in SendOpt 208
- waitExternal
 - in lccTask class 230
- waitForAID
 - in Example of terminal control 43
 - in lccTerminal class 256
- waitOnAlarm
 - in lccAlarmRequestId class 83
 - in lccTask class 230
- WaitPostType
 - in Enumerations 233
 - in lccTask class 233
- WaitPurgeability
 - in Enumerations 233
 - in lccTask class 233
- width
 - in lccTerminal class 257
- workArea
 - in lccSystem class 221
 - in lccTask class 230

- workArea (*continued*)
 - in lccTerminal class 257
- Working with lccResource subclasses
 - in Buffer objects 25
 - in lccBuf class 25
- write
 - in lccConsole class 110
- writeAndGetReply
 - in lccConsole class 111
- writeln
 - in C++ Exceptions and the Foundation Classes 51
 - in Calling methods on a resource object 21
 - in lccDataQueue class 122
 - in lccTempStore class 237, 238
 - in Temporary storage 39
 - in Transient Data 37
 - in Working with lccResource subclasses 25
 - in Writing data 38
 - in Writing items 40
- writeRecord
 - in Example of file control 31
 - in lccFile class 141
 - in lccJournal class 155
 - in Writing KSDS records 29
 - in Writing records 28
 - in Writing RRDS records 29
- writeRecord method
 - lccFile class 28
- Writing data 38
 - in Transient Data 38
 - in Using CICS Services 38
- Writing ESDS records
 - in File control 29
 - in Writing records 29
- Writing items 40
 - in Temporary storage 40
 - in Using CICS Services 40
- Writing KSDS records
 - in File control 29
 - in Writing records 29
- Writing records
 - in File control 28
 - in Using CICS Services 28
 - Writing ESDS records 29
 - Writing KSDS records 29
 - Writing RRDS records 29
- Writing RRDS records
 - in File control 29
 - in Writing records 29

X

- X
 - in actionOnConditionAsChar 184
 - in operatingSystem 220
- xldb 48
- XPLINK 7

Y

- year
 - in lccAbsTime class 80
 - in lccClock class 104
- yellow
 - in Color 258
- yesNo (parameter)
 - in setFMHContained 99

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. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

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

Other company, product, and service names may be trademarks or service marks of others.

Readers' Comments — We'd Like to Hear from You

**CICS Transaction Server for z/OS
C++ OO Class Libraries
Version 3 Release 2**

Publication No. SC34-6822-02

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your 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. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send a fax to the following number: +44-1962-816151
- Send your comments via e-mail to: idrcf@hursley.ibm.com

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address



Fold and Tape

Please do not staple

Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM United Kingdom Limited
User Technologies Department (MP095)
Hursley Park
Winchester
Hampshire
SO21 2JN
United Kingdom

Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5655-M15

SC34-6822-02



Spine information:



CICS Transaction Server for z/OS C++ OO Class Libraries

Version 3
Release 2