

CICS Transaction Gateway



Programming Reference

Version 7.1

CICS Transaction Gateway



Programming Reference

Version 7.1

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 297.

Third Edition (July 2008)

This edition applies to Version 7.1 of the CICS Transaction Gateway, program number 5724-I81, 5655-R25 and 5724-J09. It will also apply to all subsequent versions, releases, and modifications until otherwise indicated in new editions.

This edition replaces SC34-6674. Technical changes to the text are indicated by a vertical line to the left of the change.

© Copyright International Business Machines Corporation 1989, 2008.

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

Contents

About this book	vii	Methods	32
Who should read this book	vii	Session COM class	33
Installation path	vii	Interface Selection	33
Directory delimiters	vii	Object Creation	33
Information specific to your operating system	vii	Methods	34
Changes to programming reference information	ix	Terminal COM class	35
Chapter 1. COM.	1	Interface Selection	35
Buffer COM class	1	Object Creation	35
Interface Selection	1	Methods	35
Object Creation	1	UOW COM class	45
Methods	2	Interface Selection	45
Connect COM class	4	Object Creation	45
Interface Selection	4	Methods	45
Object Creation	4	Chapter 2. Java	47
Methods	5	Class/interface page	47
ECI COM class	10	Use page	48
Interface Selection	10	Tree (Class Hierarchy).	48
Object Creation	11	Index page	48
Methods	11	Chapter 3. C++.	49
EPI COM class	13	Ccl class	49
Interface Selection	13	Enumerations	49
Object Creation	13	CclBuf class	49
Methods	14	CclBuf constructors	50
Field COM class	17	Public methods	51
Interface Selection	17	Enumerations	55
Methods	17	CclConn class	56
Flow COM class	22	CclConn constructor	56
Interface Selection	22	Public methods	57
Object Creation	23	Enumerations	61
Methods	23	CclECI class	61
Map COM class	25	CclECI constructor (protected)	62
Interface Selection	26	Public methods	62
Object Creation	26	CclEPI class	64
Methods	26	CclEPI constructor	64
Screen COM class	27	Public methods	64
Interface Selection	28	Enumerations	66
Methods	28	CclException class	66
SecAttr COM class	30	Public methods	67
Interface Selection	30	CclField class	68
Methods	31	Public methods	68
SecTime COM class	32	Enumerations	72
Interface Selection	32	CclFlow class	74
		CclFlow constructor	74

Public methods	75
Enumerations	77
CclMap class	78
CclMap constructor	78
Public methods	78
Protected methods	79
CclScreen class	80
Public methods	81
Enumerations	83
CclSecAttr	83
Public Methods	83
CclSecTime	84
Public Methods	84
CclSession class	85
CclSession constructor	85
Public methods	86
Enumerations	87
CclTerminal class	87
CclTerminal constructor	87
Public methods	89
Enumerations	95
CclUOW class	97
CclUOW constructor	97
Public methods	97

Chapter 4. C and COBOL 99

External Call Interface	99
CICS_ExternalCall ECL_Parms	99
Call types for the CICS_ExternalCall	102
ECI status block	131
CICS_EciListSystems NameSpace	
Systems List	132
External Presentation Interface	133
EPI constants and data structures	133
EPI functions	139
EPI events	166
External Security Interface	170
ESI constants and data structures	170
ESI functions	172

Chapter 5. Java request monitoring, C

ECI and C EPI exits	181
Java request monitoring exits	181
C ECI exits reference	182
Identification token	183
C EPI exits reference	193
CICS_EpiInitializeExit	195
CICS_EpiTerminateExit	196
CICS_EpiAddTerminalExit	197
CICS_EpiTermIdExit	200

CICS_EpiTermIdInfoExit	201
CICS_EpiStartTranExtendedExit	202
CICS_EpiStartTranExit	203
CICS_EpiReplyExit	204
CICS_EpiDelTerminalExit	205
CICS_EpiGetEventExit	206
CICS_EpiSystemIdExit	208
CICS_EpiTranFailedExit	210

Chapter 6. Statistical API reference . . . 213

Appendix A. COM Global Constants. . . 215

Appendix B. COM EPI Specific Constants 217

Synchronization Types	217
CclEPI States	217
CclSession States	217
CclTerminal States	217
CclTerminal ATI States	218
CclTerminal EndTermReasons.	218
CclTerminal Sign-on Types.	218
CclScreen AID key codes	219
CclField Protected State Attributes	220
CclField Numeric Attributes	220
CclField Intensity Attributes	220
CclField Modified Attributes	220
CclField Highlight Attributes	221
CclField Transparency Attributes.	221
CclField Color Attributes	221

Appendix C. COM ECI Constants. . . . 223

Synchronization Types	223
Flow status types	223
Connection Status Codes	223

Appendix D. COM Error Code References 225

Appendix E. Java encodings 229

Appendix F. C++ Exception Objects . . . 233

The product library and related literature 239

CICS Transaction Gateway books	239
Sample configuration documents.	240
Redbooks	240
Other Useful Books	240
CICS Transaction Server publications	240
Microsoft Windows publications	241
APPC-related publications	241

Obtaining books from IBM.	242	Glossary	247
Accessibility features for CICS		Index	269
Transaction Gateway	243	Notices	297
Documentation	243	Trademarks	299
I Starting the Gateway daemon.	243	Sending your comments to IBM	301
Setting EPIterminal properties			
programmatically	243		
cicsterm	244		
The cicsterm -? command	245		

About this book

This information provides information about the APIs of the programming languages that the CICS® Transaction Gateway supports (Java™, C++, C, and COM) and the CICS Universal Client supports (C++, C and COM).

For information about programming methodology see the *CICS Transaction Gateway: Programming Guide*, SC34-6965-00. For further information about Java programming for the CICS Transaction Gateway, see the Javadocs shipped with this product.

Who should read this book

This book is intended for anyone involved with programming for the CICS Transaction Gateway and the CICS Universal Client.

It is assumed that you are familiar with the operating system under which your CICS Transaction Gateway or CICS Universal Client runs.

An understanding of Internet terminology is helpful.

Installation path

The term <install_path> is used in file paths to represent the directory where you installed the product. See the *CICS Transaction Gateway: Administration* book for your operating system, for the default installation locations.

Directory delimiters

References to directory path names in this book use the Microsoft® Windows® convention of a backslash (\) as delimiter, instead of the forward slash (/) delimiter used on UNIX® and Linux® operating systems.

Information specific to your operating system

Unless otherwise specified, the term *Windows* refers to Windows 2000, Windows 2003, Windows XP, and Windows Vista.

The term *Windows Terminal Server* means a server with the *Terminal Services* feature enabled.

Changes to programming reference information

- CICS Transaction Gateway V7.1 introduces IP interconnectivity (IPIC) which provides enhanced TCP/IP support for the J2EE Connector Architecture (JCA). This means that JCA now supports channels and containers, using a MappedRecord structure to hold your data.
- CICS Transaction Gateway enables users and third party vendors to write tools to access request specific information. User exit points are provided at key points in the product to allow user code to be run within the context of each individual transaction.

Chapter 1. COM

Buffer COM class

A CclOBuffer object contains a data area in memory which can be used to hold information. A particular use for a CclOBuffer object is to hold a COMMAREA used to pass data to and from a CICS server.

The CclOBuffer object is primarily intended for use with byte (binary) data. Typically a COMMAREA contains an application-specific data structure, often originating from a CICS server C program. The preferred method for handling binary data in Visual Basic is now the Byte data type. The **SetData** and **Data** methods allow the contents of the CclOBuffer object to be accessed as a Byte array. The CclOBuffer object can be used for string data, and stores strings as single-byte ANSI characters, but it does not provide any support for code-page conversions or DBCS. Note that in 32-bit environments Visual Basic uses 2-byte Unicode character representation; the COM class converts this to and from single-byte ANSI.

When a CclOBuffer object is created it allocates an area of memory as its buffer. The length of this buffer can be set explicitly via the **SetLength** method.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as CclOBuf
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Object Creation

You can create an object in two ways:

```
set var = CreateObject("Ccl.Buffer")  
set var = New CclOBuf
```

New is the preferred method in Visual Basic. For VBScript, you can use only the **CreateObject** method.

Methods

AppendString

AppendString(*string* as **String**)

string

The source string.

Appends a string to existing data in the **Ccl.Buffer** object.

Data

Data() as **Variant**

Returns the contents of the buffer as a Byte array.

ExtractString

ExtractString (*offset* as **Integer**[,
length as **Integer**]) as **String**

offset

The offset into the data area.

length

(optional) The length, in bytes, of the string to be extracted.

Returns a string from the data area starting at the specified offset.

If *length* is not specified, **ExtractString** returns data until it finds the first null terminator. If *length* is specified, **ExtractString** returns the number of bytes requested, including any nulls found in the string.

InsertString

InsertString (*offset* as **Integer**,
string as **String**)

offset

The offset in the data area where the string is to be inserted.

string

The source string.

Inserts the given string into the data area at the given offset.

Length

Length() as Integer

Returns the length of the data area in bytes.

Overlay

Overlay (*offset* as Integer, *string* as String)

offset

The offset in the data area where the string is to be inserted.

string

The source string.

Overlays the data area with the given string, starting at the given offset.

SetData

SetData(*array* as Variant)

array

The array containing the source data.

Copies the supplied array into the buffer. Byte, Integer, and Long arrays are supported.

SetLength

SetLength(*length* as Integer)

length

The new length of the data area, in bytes.

Changes the current length of the data area. If you increase the length of the buffer object, the extra space is padded with nulls. The Client daemon truncates any nulls before sending the buffer to a CICS server. If you decrease the length of the buffer object, the contents are truncated.

SetString

SetString(*string* as String)

string
Source string

Copies the supplied string into the object.

String

String() as String

Returns the contents of the **Ccl.Buffer** object as a string.

Connect COM class

The **Connect** COM class is used to maintain and represent an ECI connection between a client and a named server. Access to the server is optionally controlled by a user ID and password. It can call a program in the server or get information on the state of the connection.

Before the **Connect** COM class can be used to make calls to CICS, it must be initialized using the **Details** method and, optionally, the **TranDetails** method.

Any interaction between client and server requires a **CclOFlow** object and a **CclOConnect** object.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as Cc10Conn
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Object Creation

You can create an object in two ways:

```
set var = CreateObject("Ccl.Connect")  
set var = New Cc10Conn
```

New is the preferred method in Visual Basic. For VBScript, you can use only the **CreateObject** method.

Methods

AlterSecurity

AlterSecurity(*newUserId* as String, *newPassword* as String)

newUserId

The new userid

newPassword

The new password corresponding to the new userid.

Sets the userid and password to be used on the next link call.

Cancel

Cancel(*flow* as Object)

or

Cancel(*flow* as CclOFlow)

flow

The CclOFlow object used to control the client/server call

Cancels any **Changed** call that was previously issued to the server associated with this connection.

Changed

Changed(*flow* as Object)

or

Changed(*flow* as CclOFlow)

flow

The CclOFlow object used to control the client/server call.

Requests the server to notify the client when the current connection status changes. The call is ignored if there is an outstanding **Changed** call for this connection.

ChangePassword

ChangePassword (*newPassword* as String) as Object

or

ChangePassword (*newPassword* as String) as CclOSecAttr

newPassword

The new password

Allows a client application to change the password held in the Connect object and the password recorded by an external security manager for the *userid* held in the Connect object. The external security manager is assumed to be located in the server defined by the Connect object. A CclOSecAttr object is returned if no errors occur.

Details

Details (*serverName* as String,
userId as String,
password as String)

serverName

The name of the server. If no name is supplied the default server—the first server named in the Gateway initialization file—is used. You can discover this name, after the first call to the server by using the **ServerName** method. The length is adjusted to 8 characters by padding with blanks.

userId

The user ID, if needed. The length is adjusted to 16 characters by padding with blanks.

password

The password corresponding to the user ID in *userId*, if needed. The length is adjusted to 16 characters by padding with blanks.

Use this method to supply details of the CICS server. No interaction with the CICS server takes place until the **Link**, **Status** or **Changed** methods are called. The user ID and password are not needed if the connection is only used for status calls or if the server has no security.

Link

Link (*flow* as **Object**,
 programName as **String**,
 commArea as **Object**,
 unitOfWork as **Object**)

or

Link (*flow* as **CclOFlow**,
 programName as **String**,
 commArea as **CclOBuf**,
 unitOfWork as **CclOUOW**)

flow

The CclOFlow object used to control the client/server call.

programName

The name of the server program that is being called. The length is adjusted to 8 characters by padding with blanks or truncating, if necessary.

commArea

A CclOBuffer object that holds the data to be passed to the called program in a COMMAREA. A NULL value should be supplied if no COMMAREA is to be sent.

unitOfWork

The CclOUOW object that identifies the unit of work (UOW) with which this call is being associated. A NULL value should be supplied if no UOW is to be used.

Calls the specified program on the server. The server program sees the incoming call as an EXEC CICS LINK call.

MakeSecurityDefault

MakeSecurityDefault()

Informs the client that the current userid and password for this object is to become the default for ECI and EPI requests passed to the server as specified in the construction of the Connect object.

Password

Password() as String

Returns the password held by the CclOConnect object, padded with spaces.

ServerName

ServerName() as String

Returns the name of the server system held by the CclOConnect object and listed by the Gateway initialization file, or blanks if the default server is being used and no calls have yet been made.

ServerStatus

ServerStatus() as Integer

or

ServerStatus() as CclConnectStatusCodes

Returns the status of the server connection, set by an earlier **status** or **changed** request. Possible values are:

cclUnknowncclUnknown

The CICS server status is unknown

cclAvailablecclAvailable

The CICS server is available

cclUnavailablecclUnavailable

The CICS server is not available

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

ServerStatusText

ServerStatusText() as String

Returns a string, set by an earlier **status** or **changed** request, indicating the availability of the server.

Status

Status(*flow* as Object)

or

Status(*flow* as CclOFlow)

flow

The CclOFlow object used to control the client/server call

Request the status of the server connection.

TranDetails

TranDetails (*runTran* as String,
attachTran as String)

runTran

The CICS transaction under which called programs will run. The default is to use the default server transaction. The length is adjusted to four characters by padding with blanks.

attachTran

The CICS transaction to which called programs are attached. The default is to use the default CPMI. The length is adjusted to four characters by padding with blanks.

This method is used to supply additional information to the CICS server. The information is optional, but can be used to affect the environment in which programs are run on the CICS server.

Note: Use the Details method, to supply details of the CICS server, before using the TranDetails method; see “Details” on page 6.

UnpaddedPassword

UnpaddedPassword() as String

Returns the password held by the CclOConnect object, but with no padding with spaces at the end.

UnpaddedServerName

UnpaddedServerName() as String

Returns the server name held by the CclOConnect object, but with no padding with spaces at the end.

UnpaddedUserid

UnpaddedUserid() as String

Returns the userid held by the CclOConnect object, but with no padding with spaces at the end.

Userid

Userid() as String

Returns the user ID held by the CclOConnect object, padded with spaces, or blanks if none.

VerifyPassword

VerifyPassword() as Object

or

VerifyPassword() as CclOSecAttr

Allows a client application to verify that the password held in the Connect object matches the password recorded by an external security manager for the userid held in the Connect object. The external security manager is assumed to be located in the server defined by the Connect object. A CclOSecAttr Object is returned if no errors occur.

ECI COM class

All applications using the ECI COM class must first create a CclOECI object.

The ECI COM class provides details of candidate CICS servers. It can also be used to obtain error information.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as CclOECI
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Object Creation

You can create an object in two ways:

```
set var = CreateObject("Ccl.ECI")  
set var = New Cc10ECI
```

New is the preferred method in Visual Basic. For VBScript, you can use only the **CreateObject** method.

Methods

ErrorFormat

ErrorFormat() as Integer

Returns a value indicating the current setting for the Error Message Format. Refer to "SetErrorFormat" on page 13 for a current list of valid values.

ErrorOffset

ErrorOffset() as Long

Returns a value which can be used to convert a Client daemon error value retrieved from the ERR.Number method into the documented ExCode error values. For more information on how to do this, refer to *CICS Transaction Gateway: Programming Guide*.

ErrorWindow

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

ErrorWindow(*display* as Boolean)

display

- | | |
|--------------|---|
| true | Permits the error window to be displayed to the user. This is the default setting. |
| false | The error window will not be displayed to the user. The application must check for errors using the "ExCode" on page 12 method. |

ExCode

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

ExCode() as Integer

or

ExCode() as CclECIExceptionCodes

Returns an enumeration that indicates the last ECI error.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

The **ExCodeText** method returns a text string describing the error value.

ExCodeText

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

ExCodeText() as String

Returns a text string describing the last ECI error.

ServerCount

ServerCount() as Integer

Returns the number of candidate servers to which the client may be connected, as configured in the Gateway initialization file.

ServerDesc

ServerDesc(*index* as Integer) as String

index

The number of a connected server in the list, starting from 1

Returns the description of the *index*th server.

ServerName

ServerName(*index* as Integer) as String

index

The number of a connected server in the list, starting from 1

Returns the name of the *index*th server.

SetErrorFormat

SetErrorFormat(*format* as Integer)

format

0 Old format, provided for compatibility with earlier versions only.

1 New format, provides more information in the Visual Basic and VBScript **Err** object. This format is recommended.

This method allows you to select an error message format.

EPI COM class

The EPI COM class initializes the Client daemon EPI function. It also has methods that allow you to obtain information about CICS servers which could be used. You create a CcIOEPI object before you create CcIOTerminal objects to connect to CICS servers. The **Diagnose**, **ExCode**, and **State** methods provide information on error conditions.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as CcIOEPI
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Object Creation

You can create an object in two ways

```
set var = CreateObject("Ccl.EPI")  
set var = New CcIOEPI
```

New is the preferred method in Visual Basic. For VBScript, you can use only the **CreateObject** method.

Methods

Diagnose

Diagnose() as String

Returns a character string which holds a description of the last error.

ErrorFormat

ErrorFormat() as Integer

Returns a value indicating the current setting for the Error Message Format. Refer to “SetErrorFormat” on page 16 for a current list of valid values.

ErrorOffset

ErrorOffset() as Long

Returns a value which can be used to convert a Client daemon error value retrieved from the ERR.Number method into the documented ExCode error values. For more information on how to do this, refer to *CICS Transaction Gateway: Programming Guide*.

ErrorWindow

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

ErrorWindow(*display* as Boolean)

display

- true** Permits the error window to be displayed to the user. This is the default setting.
- false** The error window will not be displayed to the user. The application must check for errors using the **ExCode** method.

ExCode

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

ExCode() as Integer

or

ExCode() as CclEPIExceptionCodes

Returns the condition code. Possible values are:

cclSystemErrorcclSystemError

An internal Client daemon system error occurred.

cclUnknownServercclUnknownServer

There is no CICS server corresponding to the supplied *index* on **ServerDesc** or **ServerName** methods.

cclNoErrorcclNoError

The call has executed normally.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

ExCodeText

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

ExCodeText() as String

Returns a string containing descriptive text for the most recent exception.

ServerCount

ServerCount() as Integer

Returns the number of candidate servers to which the Client daemon may be connected, as configured in the Gateway initialization file.

ServerDesc

ServerDesc(*index* as Integer) as String

index

The index number of a connected server (starting from 1).

Returns a description of the selected CICS server, or a NULL string if no information is available in the Gateway initialization file for the specified server.

ServerName

ServerName(*index* as Integer) as String

index

The index number of a connected server (starting from 1).

Returns the name of the requested CICS server, or a NULL string if no information is available in the Gateway initialization file for the specified server.

SetErrorFormat

SetErrorFormat(*format* as Integer)

format

- 0 Old format, provided for compatibility with earlier versions only.
- 1 New format, provides more information in the Visual Basic and VBScript **Err** object. This format is recommended.

This method allows you to select an error message format.

State

State() as Integer

or

State() as CclEPIStates

Returns a value which indicates the state of the EPI. Possible values are:

cclActive

Initialized

cclDiscon

Terminated

cclError

Error. Refer to *CICS Transaction Gateway: Programming Guide*.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

Terminate

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

Terminate()

Terminates the Client daemon EPI in a controlled manner.

Field COM class

The **Field** COM class is used to access a single field on a 3270 screen.

CclOField objects are created and deleted when 3270 data from the CICS server is processed by a CclOScreen object.

Field objects are returned by invoking a CclOScreen object's fieldbyIndex or fieldbyPosition method. For example:

```
set var=Screen.fieldbyIndex(1)
```

Methods in this class allow field text and attributes to be read and updated. Updated fields are sent to the CICS server on the next transmission.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as CclOField
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Methods

AppendText

AppendText(*textString* as String)

textString

The text string to be appended to the field.

Appends the characters within *textString* to the end of the text already in the field.

BackgroundColor

BackgroundColor() as Integer

or

BackgroundColor() as CclColorAttributes

Returns a value which indicates the background color of the field as listed in "CclField Color Attributes" on page 221.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

BaseAttribute

BaseAttribute() as Integer

Returns the 3270 base attribute of the field.

Column

Column() as Integer

Returns the column number of the position of the start of the field on the screen, with the leftmost column being 1.

DataTag

DataTag() as Integer

or

DataTag() as CclModifiedAttributes

Returns a value which indicates whether the data in the field has been modified. Possible values are:

- cclModified
- cclUnmodified

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

ForegroundColor

ForegroundColor() as Integer

or

ForegroundColor() as CclColorAttributes

Returns a value which indicates the foreground color of the field as listed in “CclField Color Attributes” on page 221.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

Highlight

Highlight() as Integer

or

Highlight() as CclHighlightAttributes

Returns a value which indicates which type of highlight is being used as listed in “CclField Highlight Attributes” on page 221.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

InputProt

InputProt() as Integer

or

InputProt() as CclProtAttributes

Returns a value which indicates whether the field is protected. Possible values are:

- cclProtect
- cclUnprotect

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

InputType

InputType() as Integer

or

InputType() as CclNumericAttributes

Returns a value which indicates whether the field is alphanumeric or numeric. Possible values are:

- cclAlphanumeric
- cclNumeric

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

Intensity

Intensity() as Integer

or

Intensity() as CclIntensityAttributes

Returns a value which indicates whether the field is normal, intense or dark. Possible values are:

- cclDark
- cclNormal
- cclIntense

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

Length

Length() as Integer

Returns the total length of the field. This includes one byte used to store the 3270 attribute byte information; therefore the actual space for data is one less byte than the value returned by this method. See also the "TextLength" on page 22 method.

Position

Position() as Integer

Returns the position of the start of the field as an offset from the top left corner of the screen. The top row consists of positions 0 to 79; the second row, positions 80 to 159; etc.

ResetDataTag

ResetDataTag()

Resets the modified data tag (MDT) to cclUnmodified.

Row

Row() as Integer

Returns the row number of the position of the start of the field on the screen. The top row is 1.

SetBaseAttribute

SetBaseAttribute(*Attribute* as Integer)

Attribute

The value of the base 3270 attribute to be entered into the field.

Sets the 3270 base attribute.

SetExtAttribute

SetExtAttribute(*Attribute* as Integer, *Value* as Integer)

Attribute

The type of extended attribute to be set.

Value

The value of the extended attribute.

Sets the extended 3270 attribute. If an invalid 3270 attribute type or value is supplied a parameter exception is raised.

SetText

SetText(*textString* as String)

textString

The null-terminated text to be entered into the field.

Copies *textString* into the field.

Text

Text() as String

Returns the text currently held in the field.

TextLength

TextLength() as Integer

Returns the number of characters currently held in the field.

Transparency

Transparency() as Integer

or

Transparency() as CclTransparencyAttributes

Returns a value which indicates the background transparency of the field as listed in “CclField Transparency Attributes” on page 221.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

Flow COM class

A CclOFlow object is used to control ECI communications for a client/server pair.

A CclOFlow object is created for each client server interaction (call from client and response from server) and destroyed when it has been used. CclOFlow objects can be reused but an attempt to reuse a CclOFlow object that is already in use is rejected.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as CclOFlow
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Object Creation

You can create an object in two ways:

```
set var = CreateObject("Ccl.Flow")  
set var = New CclFlow
```

New is the preferred method in Visual Basic. For VBScript, you can use only the **CreateObject** method.

Methods

AbendCode

AbendCode() as String

Returns a four-character CICS transaction abend code, or spaces if no abend has occurred.

CallType

CallType() as Integer

or

CallType() as CclFlowCallTypes

Returns the type of call the flow is currently executing.current

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

CallTypeText

CallTypeText() as String

Returns the type of call the flow is currently executing as text.

Diagnose

Diagnose() as String

Returns text describing the current state of the flow object.

Flowid

Flowid() as Integer

Returns a unique identifier for this flow object.

ForceReset

ForceReset()

Makes the flow inactive and resets it. Typically, this method is used to prepare a flow object for re-use or deletion after a flow has been abandoned.

Poll

Poll(*commArea* as Object) as Boolean

or

Poll(*commArea* as CclOBuf) as Boolean

commArea

A CclOBuffer object into which the returned COMMAREA will be placed. This parameter can be set to **Nothing** if no COMMAREA should be returned.

Indicates whether a reply has been received from a deferred synchronous **Backout**, **Cancel**, **Changed**, **Commit**, **Link**, or **Status** call request. This method is only valid for deferred synchronous communications. Possible values are:

True A reply has been received.

False A reply has not been received.

SetSyncType

SetSyncType(*syncType* as Integer)

or

SetSyncType(*syncType* as CclFlowSyncTypes)

syncType

The synchronization type required for this CclOFlow object. Possible values are:

- cclSync
- cclDSynccclDSync

Sets the synchronization type required for this CclOFlow object. If cclSync is used, **link** and **status** calls using this flow block the calling program until a reply is received from CICS. If cclDSyncccclDSync is used, **link** and **status** calls using this flow return immediately to the calling program. The program can then use the **Poll** method to receive the reply from CICS later.

SetTimeout

SetTimeout(Timeout as Integer)

Sets the timeout value for the flow object for the next activation of the flow. This value can be set while a flow is active, but does not affect the current active flow.

SyncType

SyncType() as Integer

or

SyncType() as CclFlowSyncTypes

Returns the type of synchronization being used.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

Timeout

Timeout() as Integer

Returns the current timeout value set for the flow object.

Wait

Wait()

Waits for a reply from the server, blocking the client process in the meantime. This method is used when a deferred synchronous call was made, but the application now wants to wait synchronously for a reply.

Map COM class

The **Map** COM class provides validation and access to 3270 screen data using symbolic information obtained from CICS BMS maps. To use this interface, run the CICSBMSC utility on your server program BMS maps.

Note: CICSBMSC is not provided with CICS Transaction Gateway for the Linux operating system. If you require this functionality, contact your local IBM® support representative and ask them to forward your request to the CICS service team.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as Cc10Map
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Object Creation

You can create an object in two ways:

```
set var = CreateObject("Ccl.Map")  
set var = New Cc10Map
```

New is the preferred method in Visual Basic. For VBScript, you can use only the **CreateObject** method.

Methods

ExCode

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

ExCode() as Integer

or

ExCode() as CcIEPIExceptionCodes

Returns a value that indicates the current condition code.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

FieldByName

FieldByName(*name* as Integer) as Object

or

FieldByName(*name* as Integer) as CclOField

name

Symbolic value for the required field. This value is provided in the <mapname>.BAS file generated from the source BMS by the CICSBMSC utility.

Returns the specified CclOField object.

Validate

Validate (*screenRef* as Object, *mapname* as String) as Boolean

or

Validate (*screenRef* as CclOScreen, *mapname* as String) as Boolean

screenRef

CclOScreen object

mapname

String value supplied in <mapname>.BAS file generated from the source BMS by the CICSBMSC utility.

Validate map against the current screen.

This method can be used to verify that a specific BMS map has been received from the CICS server. Possible return values are:

TRUE

Specified BMS map matches current screen contents.

FALSE

Specified BMS map does not match current screen contents

If TRUE is returned, the **FieldByName** method can be used to access fields using their BMS name.

Screen COM class

The **Screen COM** class maintains all data on the 3270 virtual screen and provides access to this data. It contains a collection of CclOField objects which represent the fields on the current 3270 screen.

A single Screen object is created by the Terminal object when the terminal is installed either with the Ccl Terminal connect or install method. The application gets access to the CclOScreen object via the Ccl Terminal **Screen** method.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as CclOScreen
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Methods

CursorCol

CursorCol() as Integer

Returns the current cursor column (the left col is 1).

CursorRow

CursorRow() as Integer

Returns the current cursor row (the top row is 1).

Depth

Depth() as Integer

Returns the number of rows on the screen.

FieldByIndex

FieldByIndex(*index* as Integer) as Object

or

FieldByIndex(*index* as Integer) as CclOField

index

The index number of the field required. The first field is number 1.

FieldByPosition

FieldByPosition (*rowPos* as Integer, *colPos* as Integer) as Object

or

FieldByPosition (*rowPos* as Integer, *colPos* as Integer) as CclOField

rowPos

The row number of the field (topmost row = 1).

colPos

The column number of the field (leftmost column = 1).

FieldCount

FieldCount() as Integer

Returns the number of fields on the screen.

MapName

MapName() as String

Returns a string specifying the name of the map that was most recently referenced in the MAP option of a SEND MAP command processed for the terminal resource. If the terminal resource is not supported by BMS, or the server has no record of any map being sent, the value returned is blank.

MapSetName

MapSetName() as String

Returns a string specifying the name of the mapset that was most recently referenced in the MAPSET option of a SEND MAP command processed for the terminal resource. If the MAPSET option was not specified on the most recent request, BMS used the map name as the mapset name. In both cases, the mapset name used may have been suffixed by a terminal suffix. If the terminal resource is not supported by BMS, or the server has no record of any mapset being sent, the value returned is blank.

SetAID

SetAID(*key* as Integer)

or

SetAID(*key* as CclADIKeys)

key

The AID key value as listed in “CclScreen AID key codes” on page 219.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

Sets the AID key value to be passed to the server on the next transmission.

SetCursor

SetCursor (*rowPos* as Integer, *colPos* as Integer)

rowPos

The required row number of the cursor (the top row is 1).

colPos

The required column number of the cursor (the left column is 1).

Width

Width() as Integer

Returns the number of columns on the screen.

SecAttr COM class

The **SecAttr** COM class provides information about passwords reported back by the external security manager when issuing `verifySecurity` or `changePassword` methods on `CclOConnect` or `CclOTerminal` objects.

This object is created and owned by the `CclOConnect` or `CclOTerminal` Object and access to this object is provided when invoking the `VerifyPassword` or `ChangePassword` methods.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as Cc10SecAttr
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Methods

ExpiryTime

ExpiryTime() as Object

or

ExpiryTime() as CclOSecTime

Returns a CclOSecTime object that contains the date and time when the password will expire.

InvalidCount

InvalidCount() as Integer

Returns the number of times an invalid password has been entered for the userid.

LastAccessTime

LastAccessTime() as Object

or

LastAccessTime() as CclOSecTime

Returns a CclOSecTime object which contains the date and time when the userid was last accessed.

LastVerifiedTime

LastVerifiedTime() as Object

or

LastVerifiedTime() as CclOSecTime

Returns a CclOSecTime object which contains the date and time of the last verification.

SecTime COM class

The **SecTime** COM class provides date and time information in the **CclOSecAttr** object for various entries reported back by the external security manager when issuing **verifySecurity** or **changePassword** methods on **Connect** or **Terminal** objects. These objects are created and owned by the **CclOSecAttr** object and access is obtained via the various methods available on this object. No constructors or destructors are available.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as CclOSecTime
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Methods

Day

unsigned short Day() as Integer

Returns the day in the range 1 to 31.

GetDate

GetDate() as Date

Returns the date and time in a Visual Basic DATE type format.

Hours

unsigned short Hours() as Integer

Returns the hours in the range 0 to 23.

Hundredths

unsigned short Hundredths() as Integer

Returns the hundredths of a second in the range 0 to 99.

Minutes

unsigned short Minutes() as Integer

Returns the minutes in the range 0 to 59.

Month

unsigned short Month() as Integer

Returns the month in the range 1 to 12.

Seconds

unsigned short Seconds() as Integer

Returns the seconds in the range 0 to 59.

Year

unsigned short Year() as Integer

Returns a 4 digit year.

Session COM class

The **Session** COM class controls the flow of data to and from CICS within a single EPI session.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as Cc10Session
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Object Creation

You can create an object in two ways:

```
set var = CreateObject("Ccl.Session")  
set var = New Cc10Session
```

New is the preferred method in Visual Basic. For VBScript, you can use only the **CreateObject** method.

Methods

Diagnose

Diagnose() as String

Returns the text description of the current state of the session.

SetSyncType

SetSyncType(syncType as Integer)

or

SetSyncType(syncType as CclFlowSyncTypes)

syncType

The synchronization type required for this CclOSession object. Possible values are:

- cclSync
- cclDSyncccclDSync

Sets the synchronization type required for this CclOSession object. If cclSync is used, **Start** and **Send** calls using this flow will block the calling program until a reply is received from CICS. If cclDSyncccclDSync is used, **Start** and **Send** calls using this flow will return immediately to the calling program. The program can then use the **Poll** method to receive the reply from CICS at a later time.

State

State() as Integer

or

State() as CclEPIStates

Returns a value which indicates the current state of the session. Possible values are:

cclActive

Connected

cclServer

Transaction in progress in the CICS server.

cclClient

CICS server is waiting for a response from the client

cclDiscon

Disconnected

cclError

Error, call **ExCode** and **Diagnose** methods for further information.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

TransId

TransId() as String

Returns the four-letter name of the current transaction.

Terminal COM class

The **Terminal** COM class represents a 3270 terminal connection to a CICS server. A CICS connection is established when the **Connect** method is called. Methods can then be used to converse with a 3270 terminal application (often a BMS application) in the CICS server.

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as Cc10Terminal
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Object Creation

You can create an object in two ways:

```
set var = CreateObject("Ccl.Terminal")  
set var = New Cc10Terminal
```

New is the preferred method in Visual Basic. For VBScript, you can use only the **CreateObject** method.

Methods

AlterSecurity

AlterSecurity(*newuserid* as **String**,*newpassword* as **String**)

newpassword

The new password to be given to *newuserid*.

newUserId

The new userid.

Allows you to redefine the userid and password for a terminal resource that has been constructed without them (a sign-on incapable terminal). The method can be called before you install a terminal. It changes the terminal definition; the new userid and password are be used for the terminal when install is called.

CCSId

CCSId() as long

Returns a long showing the selected code page.

ChangePassword

ChangePassword(*newPassword* as String) as Object

or

ChangePassword(*newPassword* as String) as CclOSecAttr

newPassword

The new password to be given

Allows a client application to change the password held in the terminal object and the password recorded by an external security manager for the userid held in the terminal object. The external security manager is assumed to be located in the server defined by the terminal object. A CclOSecAttr Object is returned if no errors occurred.

Connect

Connect(*servName* as String,
devType as String,
nworkName as String)

servName

The name of the server with which you want to communicate. If a NULL string is provided, the default server system, defined in the Gateway initialization file, is assumed. The name is expanded to 8 characters by padding with blanks, if necessary.

devType

The name of the model terminal definition which the server uses to

generate a terminal resource definition. If a NULL string is provided the default model is used. The name is expanded to 16 characters by padding with blanks, if necessary.

nworkName

The name of the terminal resource to be installed or reserved. The name is expanded to 8 characters by padding with blanks, if necessary. If a NULL string is supplied, the CICS server allocates a name.

Establishes a 3270 communication to the specified CICS server.

Devtype

Devtype() as String

Returns the terminal device type as a string.

Diagnose

Diagnose() as String

Returns a character string which holds a description of the error returned by the most recent server call.

Disconnect

Disconnect()

Disconnects the terminal from CICS. No attempt is made to purge any outstanding running transaction.

DisconnectWithPurge

DisconnectWithPurge()

Disconnects the terminal from CICS and attempts to purge any outstanding running transaction. This purge function does not cancel ATI requests queued against the terminal.

DiscReason

DiscReason() as CclEndTermReasons

This method will return an enumeration showing the reason the terminal has been disconnected. Possible values are shown in "CclTerminal EndTermReasons" on page 218.

ExCode

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

ExCode() as Integer

or

ExCode() as CclEPIExceptionCodes

Returns a value which indicates the most recent condition code returned by the server.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

ExCodeText

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

ExCodeText() as String

Returns a text string describing the most recent condition code returned by the server.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

Install

Install(*session* as Object, *timeout* as Integer)

or

Install(*session* as CclOSession, *timeout* as Integer)

session

The session object to be used by this terminal object.

InstallTimeout

A value in the range 0 through 3600, specifying the maximum time in seconds that installation of the terminal resource is allowed to take. A value of 0 means that no limit is set.

This method installs a non-connected terminal resource. A `cclInvalidState` error is raised if the terminal is already installed.

MakeSecurityDefault

MakeSecurityDefault()

Informs the client that the current userid and password for this object is to become the default for ECI and EPI requests passed to the server as specified in the construction of the Terminal object.

NetName

NetName() as String

Returns the network name of the terminal.

Password

Password() as String

Returns a text string containing the current password for the userid associated with the terminal. The string is empty if there is no password.

Poll

Poll() as Boolean

Checks to see if a replies have been received from a deferred synchronous **Start** or **Send** request. Possible values are:

True No further replies outstanding
False Further replies outstanding

A CICS server transaction may send more than one reply in response to a **Terminal.Start** or **Terminal.Send** call. More than one **Terminal.Poll** call may therefore be needed to collect all the replies. The return code indicates whether you need to perform more poll requests.

PollForReply

PollForReply() as Boolean

Checks to see if replies have been received from a deferred synchronous Start or Send request. Possible values are

true Replies have been received

false No replies have been received

A CICS server transaction may send more than one reply in response to a Terminal.Start or Terminal.Send call. More than one Terminal.PollForReply call may therefore be needed to collect all replies. Use the Terminal.State method to find out if further replies are expected. If there are, the value returned will be cclServer.

QueryATI

QueryATI() as Integer

or

QueryATI() as CclATISStates

Returns a value that indicates whether Automatic Transaction Initiation (ATI) is enabled or disabled. Possible values are:

- cclATIEnabled
- cclATIDisabled

ReadTimeout

ReadTimeout() as Integer

Returns the read timeout setting for the terminal.

ReceiveATI

ReceiveATI (*session* as Object)

or

ReceiveATI (*session* as CclOSession)

session

A pointer to the CclOSession object which is to be used for the CICS server interaction.

Waits for and receives 3270 data stream for a CICS ATI transaction. The CclOSession object supplied can only be synchronous.

Screen

Screen() as Object

Returns the CclOScreen object that is handling the 3270 screen associated with this terminal.

Send

Send(*session* as Object)

or

Send(*session* as CclOSession)

session

The CclOSession object which controls the session which is to be used. It is set to NULL if no CclOSession object is used.

Generates a 3270 data stream from the current contents of the **CclOScreen** object and transmits it to the CICS server.

ServerName

ServerName() as String

Returns the name of the server system held by the CclOTerminal object and listed by the Gateway initialization file, or blanks if the default server is being used and no calls have yet been made.

SetATI

SetATI(*stateVal* as Integer)

or

SetATI(*stateVal* as CclATISStates)

stateVal

A value which indicates whether the ATI is to be enabled or disabled. Possible values are:

- cclATIEnabled

- cclATIDisabled

SetTermDefns

SetTermDefns (*servName* as String,
devType as String,
nworkName as String
signonCapability as CclSignonTypes
userid as String
password as String
ReadTimeout as Integer
CCSid as Long)

servName

The name of the server with which you want to communicate. If a NULL string is provided, the default server system, defined in the Gateway initialization file, is assumed. The name is expanded to 8 characters by padding with blanks, if necessary.

devType

The name of the model terminal definition which the server uses to generate a terminal resource definition. If a NULL string is provided the default model is used. The name is expanded to 16 characters by padding with blanks, if necessary.

nworkName

The name of the terminal resource to be installed or reserved. The name is expanded to 8 characters by padding with blanks, if necessary. If a NULL string is supplied, the CICS server will allocate a name.

signonCapability

Set the sign-on capability to one of the following:

cclSignonCapable

cclSignonIncapable

ReadTimeout

A value in the range 0 through 3600, specifying the maximum time in seconds between the time the classes go clientrepl state and the time that the application program invokes the reply method.

userid

The name of the userid to associate with this terminal resource.

password

The password to associate with the userid.

CCSid

A long specifying the coded character set identifier (CCSID) that identifies

the coded graphic character set used by the client application for data passed between the terminal resource and CICS transactions. A zero indicates that a default will be used.

Creates a terminal resource but does not make the connection to the Server.

SignonCapability

SignonCapability() as Integer

or

SignonCapability() as CclSignonTypes

Returns the type of terminal installed. Possible values are:

- cclSignonCapable
- cclSignonIncapable

Start

Start (*session* as Object,
tranCode as String,
startData as String)

or

Start (*session* as CclOSession,
tranCode as String,
startData as String)

session

The CclOSession object which controls the session which is to be used. It is set to NULL if no CclOSession object is used.

tranCode

The name of the transaction which is to be started

startData

Start transaction data. A NULL value indicates no data is required for the transaction being started.

Generates a 3270 data stream from the supplied data and transmits it to the CICS server, starting the named transaction.

State

State() as Integer

or

State() as CclEPIStates

Returns a value which indicates the current state of the session. These values are the same as those returned by the **state** method in the **Session** COM class.

Constants are available in the type library. Use the Visual Basic Object Browser to view them.

TermId

TermId() as String

Returns the terminal ID.

TransId

TransId() as String

Returns the 4-character name of the current CICS transaction. Note that if a RETURN IMMEDIATE is run from the current transaction, TransId does not provide the name of the new transaction; it still contains the name of the first transaction.

Userid

Userid() as String

Returns a text string containing the current userid for the terminal. The string is empty if there is no userid.

VerifyPassword

VerifyPassword() as Object

or

VerifyPassword() as CclOSecAttr

Allows a client application to verify that the password held in the terminal object matches the password recorded by an external security manager for the userid held in the terminal object. The external security manager is assumed

to be located in the server defined by the terminal object. A CclOSecAttr Object is returned if no errors occurred.

UOW COM class

Use this COM class when you make updates to recoverable resources in the server within a “unit of work” (UOW). Each update in a UOW is identified by a reference to its CclOUOW object — see **Link** method in **Connect** COM class (“Link” on page 6).

Interface Selection

For Visual Basic, the following types of interface are available:

```
Dim var as Object  
Dim var as CclOUOW
```

The second method is preferred.

If you do not dim a variable, dim it with no type, or are using VBScript, the variable is assumed to be of type **Object**.

Object Creation

You can create an object in two ways:

```
set var = CreateObject("Ccl.UOW")  
set var = New CclOUOW
```

New is the preferred method in Visual Basic. For VBScript, you can use only the **CreateObject** method.

Methods

BackOut

BackOut(*flow* as Object)

or

BackOut(*flow* as CclOFlow)

flow

The CclOFlow object which is used to control the client/server call

Terminate this UOW and back out all changes made to recoverable resources in the server.

Commit

Commit(*flow* as Object)

or

Commit(*flow* as CclOFlow)

flow

The CclOFlow object which is used to control the client/server call

Terminate this UOW and commit all changes made to recoverable resources in the server.

ForceReset

ForceReset()

Makes this UOW inactive and resets it. The UOW is neither committed or backed out.

UowId

UowId() as long

Returns the identifier of the UOW. A zero return indicates that the UOW is either complete or has not yet started, in other words it is inactive.

Chapter 2. Java

Online programming reference information is provided for the Java classes and interfaces provided with CICS Transaction Gateway.

The reference information is in HTML format and is generated using the Javadoc tool provided with the JDK.

The following sections describe the different kinds of HTML pages that are provided within the reference information.

See the README file for the latest information on using the programming reference information.

Class/interface page

In the reference pages, each class and interface has its own page. In each of these pages, there are three sections:

1. Class/interface description:
 - Class inheritance diagram
 - Direct Subclasses
 - All Known Subinterfaces
 - All Known Implementing Classes
 - Class/interface declaration
 - Class/interface description
2. Summary tables:
 - Inner Class Summary
 - Field Summary
 - Constructor Summary
 - Method Summary
3. Class/interface description:
 - Field Detail
 - Constructor Detail
 - Method Detail

Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Use page

Each documented class and interface has its own Use page. This page describes what packages, classes, methods, constructors and fields use any part of the given class. The Use page for a package or interface A includes:

- Subclasses of A
- Fields declared as A
- Methods that return A
- Methods and constructors with parameters of type A.

To access this page, go to the class or interface, then click on the **Use** link in the navigation bar.

Tree (Class Hierarchy)

When viewing a particular class or interface page, selecting **Tree** displays the class and interface hierarchy for CICS Transaction Gateway.

Index page

The Index contains an alphabetic list of all classes, interfaces, constructors, methods, and fields.

Related information

Request monitoring user exit API information

Chapter 3. C++

Ccl class

This class defines enumerations which are used by other classes—both ECI and EPI.

Enumerations

Bool

There are two equivalent pairs of values:

- no and yes
- off and on

Sync

Possible values are:

async asynchronous
dsync deferred synchronous
sync synchronous

ExCode

For possible values, refer to Table 22 on page 233.

CclBuf class

A CclBuf object contains a data area in memory that can be used to hold information. A particular use for a CclBuf object is to hold a COMMAREA, which passes data to and from a CICS server.

The CclBuf object is primarily intended for use with byte (binary) data. A typical COMMAREA contains an application-specific data structure, often originating from a CICS server PL/1 or C program. Methods such as **assign()** and **insert()** therefore provide a **void*** parameter type for application data input. There is limited support for SBCS null-terminated strings (some of the code samples make use of this), but there is no code-page conversion or DBCS support in the **CclBuf** class.

The maximum data length for a buffer is the maximum value for unsigned long (2^{32}) for 32 bit platforms. CICS imposes a limit of 32 KB in

COMMAREAs. This may be reduced by setting the *MaxBufferSize* parameter in the CICS Transaction Gateway initialization file. See the *CICS Transaction Gateway: Administration* book for your operating system, for more information. If a buffer object used as a COMMAREA is too long, a data length exception is raised.

When a `CclBuf` object is created it either uses an area of memory passed to it as its buffer, or allocates its own. The length of the data in this buffer can be reduced after the `CclBuf` object is created. The length of the data in this buffer can only be increased beyond the original length if the `CclBuf` object is created with a **DataAreaType** of extensible, rather than fixed.

If a buffer object has a **DataAreaType** of fixed and a method is called which would result in its data area length being exceeded, a buffer overflow exception is raised. If the exception is not handled, the buffer will contain the result of the call, truncated to the data area length.

If a method is called that results in a buffer object having a data length smaller than its data area length, the data is padded with nulls.

Many of the methods return object references. This makes it possible for users to chain calls to member functions. For example, the code:

```
CclBuf comm1;  
comm1="Some text";  
comm1.insert( 9,"inserted ",5) += " at the end";
```

would create the following string:

Some inserted text at the end

CclBuf constructors

CclBuf (1)

CclBuf(unsigned long *length*, DataAreaType *type* = extensible)

length

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

type

An enumeration indicating whether the data area can be extended.

Possible values are extensible or fixed. The default is extensible.

Creates a `CclBuf` object, allocating its own data area with the given length. All the bytes within it are set to null. The data length is set to zero and remains zero until data is put in the buffer.

CclBuf (2)

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

length

The length of the supplied data area, in bytes.

dataArea

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

Creates a CclBuf object that cannot be extended, adopting the given data area as its own. The DataAreaOwner is set external.

CclBuf (3)

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

text

A string to be copied into the new CclBuf object.

type

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

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

CclBuf (4)

CclBuf(const CclBuf& *buffer*)

buffer

A reference to the CclBuf object that is to be copied.

This copy constructor creates a new CclBuf object, which is a copy of the given object. The data length, data area length and data area type of the new buffer are the same as the old buffer. The data area owner of the new buffer is internal.

Public methods

assign

CclBuf& 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.

Overwrites the current contents of the data area with the source data and resets the data length.

cut

CclBuf& 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 zero.

Cuts the specified data from the data area. Data in the data area is padded with nulls.

dataArea

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

offset

The offset into the data area. The default is zero.

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

dataAreaLength

unsigned long dataAreaLength() const

Returns the length of the data area in bytes.

dataAreaOwner

DataAreaOwner dataAreaOwner() const

Returns an enumeration value indicating whether the data area has been allocated by the **CclBuf** constructor or has been supplied from elsewhere. Possible values are internal and external.

dataAreaType

DataAreaType dataAreaType() const

Returns an enumeration value indicating whether the data area can be extended. Possible values are extensible and fixed.

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

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

length

The length of the data, in bytes, to be inserted into the CclBuf object.

dataArea

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

offset

The offset into the data area where the data is to be inserted. The default is zero.

Inserts the source data into the data area at the given offset.

listState

const char* listState() const

Returns a formatted string containing the current state of the object. For example:

```
Buffer state..&CclBuf=000489B4 &CclBufI=00203A00  
dataLength=8 &dataArea=002039C0  
dataAreaLength=8 dataAreaOwner=0 dataAreaType=1
```

operator= (1)

CclBuf& operator=(const CclBuf& buffer)

buffer

A reference to a CclBuf object.

Assigns data from another buffer object.

operator= (2)

CclBuf& operator=(const char* text)

text

The string to be assigned to the CclBuf object.

Assigns data from a string.

operator+= (1)

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

buffer

A reference to a CclBuf object.

Appends data from another buffer object to the data in the data area.

operator+= (2)

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

text

The string to be appended to the CclBuf object.

Appends a string to the data in the data area.

operator==

Ccl::Bool operator==(const CclBuf& *buffer*) const

buffer

A reference to a CclBuf object.

Returns an enumeration indicating whether the data contained in the buffers of the two CclBuf objects is the same. Possible values are yes, indicating that the data lengths and contents are the same, or no.

operator!=

Ccl::Bool operator!=(const CclBuf& *buffer*) const

buffer

A reference to a CclBuf object.

Returns an enumeration indicating whether the data contained in the buffers of the two CclBuf objects is different. Possible values are yes or no. no means that the data lengths are the same and the contents are the same.

replace

```
CclBuf& 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 **CclBuf** data area. The default is zero.

Overwrites the current contents of the data area at the given offset with the source data. The data length remains the same.

setDataLength

```
unsigned long setDataLength(unsigned long length)
```

length

The new length of the data area, in bytes.

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

If *length* is greater than the data area length, the data is padded with nulls.

Enumerations

DataAreaOwner

Indicates whether the data area of a **CclBuf** object has been allocated outside the object. Possible values are:

internal

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

external

The data area has been allocated externally.

DataAreaType

Indicates whether the data area of a **CclBuf** object can be made longer than its original length. Possible values are:

extensible

The data area of a CclBuf object can be made longer than its original length.

fixed The data area of a CclBuf object cannot be made longer than its original length.

CclConn class

An object of class **CclConn** is used to represent an ECI connection between a client and a named server. See *Server connection* in the *CICS Transaction Gateway: Programming Guide*. Access to the server is optionally controlled by a `userId` and password. It can call a program in the server or get information on the state of the connection. See *Passing data to a server program* in the *CICS Transaction Gateway: Programming Guide* and *Monitoring server availability* in the *CICS Transaction Gateway: Programming Guide* for more information.

The creation of a CclConn object does not cause any interaction with the CICS server, nor does it guarantee that the server is available to process requests.

Any interaction between client and server requires the use of a CclFlow object. See *Compiling and Linking* in the *CICS Transaction Gateway: Programming Guide* for more information.

A CclConn object cannot be copied or assigned. Any attempt to delete a CclConn object for which there are active CclFlow or CclUOW objects raises an `activeFlow` or an `activeUOW` exception.

CclConn constructor

```
CclConn(const char* serverName = 0,  
        const char* userId = 0,  
        const char* password = 0,  
        const char* runTran = 0,  
        const char* attachTran = 0)
```

serverName

The name of the server. If no name is supplied the default server is used. After the first call to the server you can discover this name by using the `serverName` method. The length is adjusted to 8 characters by padding with blanks or truncating, if necessary.

userId

The `userId`, if needed. The length is adjusted to 16 characters by padding with blanks or truncating, if necessary.

password

The password corresponding to the *userId* in *userID*, if needed. The length is adjusted to 16 characters by padding with blanks or truncating, if necessary.

runTran

The CICS transaction under which the called program will run. The default is to use the default server transaction. The length is adjusted to 4 characters by padding with blanks or truncating, if necessary.

attachTran

The CICS transaction to which the called program is attached. The default is to use the default CPML. The length is adjusted to 4 characters by padding with blanks or truncating, if necessary.

This constructor creates a *CclConn* object; it does not cause any interaction with the CICS server or guarantee that the server is available to process requests. The *userId* and *password* are not needed if the connection is used only for status calls, or if the server has no security.

Public methods

alterSecurity

void alterSecurity(const char* *newUserId*, const char* *newPassword*)

newUserId

The new *userid*

newPassword

The new password corresponding to the new *userid*

Updates the *UserId* and *Password* to be used on the next link call

cancel

void cancel(CclFlow& *flow*)

flow

A reference to the *CclFlow* object used to control the server request call.

Cancels any **changed** call that was previously issued to the server associated with this connection.

changed

void changed(CclFlow& *flow*)

flow

A reference to the *CclFlow* object used to control the server request call.

Requests the server to notify the Client daemon when the current connection status changes. The call is ignored if there is already an outstanding **changed** call for this connection. Use **serverStatus** or **serverStatusText** to obtain server availability.

changePassword

CclSecAttr* changePassword(const char* newPassword)

newPassword

the new password to be given

Allows a Client application to change:

- The password held in the terminal object
- The password recorded by an external security manager for the userid held in the terminal object

The external security manager is assumed to be located in the server defined by the terminal object.

link

```
void link(CclFlow& flow,  
         const char* programName,  
         CclBuf* commarea = 0,  
         CclUOW* unit = 0)
```

flow

A reference to the CclFlow object used to control the server request call.

programName

The name of the server program that is being called. The length is adjusted to 8 characters by padding with blanks or truncating, if necessary.

commarea

A pointer to a CclBuf object that holds the data to be passed to the called program in a COMMAREA. The default is not to pass a COMMAREA.

unit

A pointer to the CclUOW object that identifies the unit of work (UOW) in which this call participates. The default is none. See *Managing logical units of work* in the *CICS Transaction Gateway: Programming Guide*.

Requests execution of the specified program on the server. The server program sees the incoming call as an EXEC CICS LINK call.

If the *commarea* buffer object is too long, a *dataLength* exception is raised and the request is denied. CICS imposes a limit of 32 KB which can be made

smaller by using the `MaxBufferSize` parameter in the CICS Transaction Gateway Initialization file.

listState

const char* listState() const

Returns a formatted string containing the current state of the object. For example:

```
Connection state..&CclConn=000489AC &CclConnI=00203A50
flowCount=0 &CclFlow(changed)=00000000 token(changed)=0
serverName="server " userId="userId " password="password "
&CclUOWI=00000000 runTran="run " attachTran="att "
```

makeSecurityDefault

void makeSecurityDefault()

Informs the client that the current userid and password for this object is to become the default for ECI and EPI requests passed to the server as specified in the construction of the connection object.

password (1)

const char* password() const

Returns the password held by the `CclConn` object, padded with spaces to 10 characters, or blanks if there is no password.

password (2)

void password(Ccl::Bool *unpadded*)

unpadded

Ccl::Yes

returns a null terminated string of the stored password with no space padding in the string.

Ccl::No

returns the string padded with spaces — the same as invoking the `password` method with no parameters.

serverName (1)

const char* serverName() const

Returns the name of the server system held by the **CclConn** object, padded with spaces, or blanks if the default server is being used and no calls have yet been made.

serverName (2)

void serverName(Ccl::Bool *unpadded*)

unpadded

Ccl::Yes

returns a null terminated string of the stored server name with no space padding in the string.

Ccl::No

returns the string padded with spaces — the same as invoking the **serverName** method with no parameters.

status

void status(CclFlow& *flow*)

flow

A reference to the **CclFlow** object used to control the server request call.

Requests the status of the server connection. When the reply has been received, use **serverStatus** or **serverStatusText** to obtain server availability.

serverStatus

ServerStatus serverStatus() const

Returns an enumeration value, set by an earlier **status** or **changed** request, indicating the availability of the server. Possible values are listed under Enumerations.

serverStatusText

const char* serverStatusText() const

Returns a string, set by an earlier **status** or **changed** request, indicating the availability of the server.

userId (1)

const char* userId() const

Returns the user ID held by the **CclConn** object, padded with spaces, or blanks if none.

userId (2)

void **userId**(**Ccl::Bool** *unpadded*)

unpadded

Ccl::Yes

returns a null terminated string of the stored userid with no space padding in the string.

Ccl::No

returns the string padded with spaces exactly as invoking the **userId** method with no parameters.

verifyPassword

CclSecAttr* **verifyPassword**()

Allows a Client application to verify that the password held in the **CclConn** object matches the password recorded by an external security manager for the **userid** held in the **CclConn** object. The external security manager is assumed to be located in the server defined by the **CclConn** object.

Enumerations

ServerStatus

Indicates the availability of the server. Possible values are:

unknown

The server status is unknown.

available

The server is available.

unavailable

The server is not available.

CclIECI class

Only one instance of the **CclIECI** class can exist. It is created by the **instance** class method. It controls the client interface to the available servers.

CclIECI should be sub-classed to implement your own **handleException** method.

Only one instance of a **CclIECI** subclass can exist. Any attempt to create more than one raises a **multipleInstance** exception.

A **CclIECI** object cannot be copied or assigned.

CclECI constructor (protected)

CclECI()

This constructor is protected and can be accessed only from a subclass.

Public methods

exCode

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

Ccl::ExCode exCode() const

Returns an enumeration indicating the most recent exception code. The possible values are listed under Table 22 on page 233.

exCodeText

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

const char* exCodeText() const

Returns a text string describing the most recent exception code.

handleException

virtual void handleException(CclException &except)

except

A CclException object that contains information about the exception just raised.

This method is called whenever an exception is raised. To deal with exceptions, you should always subclass **CclECI**, and provide your own implementation of **handleException**. See *Handling Exceptions* in the *CICS Transaction Gateway: Programming Guide*. The default implementation merely throws the exception object.

instance

static CclECI* instance()

A class method that returns a pointer to the single CclECI object that exists on the client. Here is an example of its use:

```
CclECI* pmgr = CclECI::instance();
```

listState

const char* listState() const

Returns a formatted string containing the current state of the object. For example:

```
ECI state..&CclECI=00203AE0 &CclECII=00203B20  
retCode=0 exCode=0  
serverCount=0 &serverBuffer=00000000
```

serverCount

unsigned short serverCount() const

Returns the number of available servers to which the CICS Transaction Gateway may be connected, as configured in the CICS Transaction Gateway initialization file. In practice, some or all of these servers may not be available. See *Finding potential servers* in the *CICS Transaction Gateway: Programming Guide*.

serverDesc

const char* serverDesc(unsigned short *index* = 1) const

index

The index of a connected server in the list. The default index is 1.

Returns the description of the *index*th server. See *Finding potential servers* in the *CICS Transaction Gateway: Programming Guide*.

serverName

const char* serverName(unsigned short *index* = 1) const

index

The index of a connected server in the list. The default index is 1.

Returns the name of the *index*th server. See *Finding potential servers* in the *CICS Transaction Gateway: Programming Guide*.

CcIEPI class

The **CcIEPI** class initializes and terminates the CICS Transaction Gateway EPI function. It also has methods which allow you to obtain information about CICS servers configured in the CICS Transaction Gateway initialization file. You must create one object of this class for each application process before you create **CclTerminal** objects to connect to CICS servers.

CcIEPI constructor

CcIEPI()

This method initializes the CICS EPI interface on the client. An **initEPI** exception is raised if initialization fails. Initialization of the CICS Transaction Gateway EPI is synchronous. In other words, initialization is complete when the call to the **CcIEPI** constructor returns.

Public methods

diagnose

const char* diagnose() const

Returns a character string that holds a description of the condition returned by the most recent server call.

exCode

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

Ccl::ExCode exCode() const

Returns an enumeration indicating the most recent exception code. The possible values are listed under Table 22 on page 233.

exCodeText

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

const char* exCodeText() const

Returns a text string describing the most recent exception code.

handleException

virtual void handleException(CclException &except)

except

A CclException object that contains information about the exception just raised.

This method is called whenever an exception is raised. To deal with exceptions, use *try...catch*, or subclass **CclEPI** and provide your own implementation of **handleException**. The default implementation merely throws the exception object.

serverCount

unsigned short serverCount()

Returns the number of available servers to which the CICS Transaction Gateway may be connected, as configured in the CICS Transaction Gateway initialization file.

serverDesc

const char* serverDesc(unsigned short index = 1)

index

The index of a configured server

Returns a description of the selected CICS server, or NULL if no information is available in the CICS Transaction Gateway initialization file for the specified server. If the index exceeds the number of servers configured, a `maxServers` exception is raised.

serverName

const char* serverName(unsigned short index = 1)

index

The index of a configured server

Returns the name of the requested CICS server, or NULL if no information is available in the CICS Transaction Gateway initialization file for the specified server. If the index exceeds the number of servers configured, a `maxServers` exception is raised.

state

State state() const

Returns an enumeration indicating the state of the EPI. Possible values are:

active EPI has been initialized successfully

discon

EPI has terminated

error EPI initialization has failed

terminate

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

void terminate()

Terminates the CICS Transaction Gateway EPI in a controlled manner. The CclEPI object remains in existence, so that anything which occurs during the termination can be monitored by the application.

Because the terminate method is invoked during CclEPI object destruction, you do not need to invoke this method.

Enumerations

State

An enumeration indicating the state of the EPI. Possible values are:

active EPI has been initialized successfully

discon

EPI has terminated

error EPI initialization has failed

CclException class

A CICS Transaction Gateway object constructs an object of the **CclException** class if it encounters a problem.

To deal with such a problem, you should subclass the **CclECI** or **CclEPI** class and provide your own implementation of the **handleException** method. See *Handling Exceptions* in the *CICS Transaction Gateway: Programming Guide*. This method has access to the methods of the CclException object and can be coded to take whatever action is necessary. For example, it can stop the program or display a dialog box.

Alternatively, you can use a C++ *try...catch* block to handle exceptions.

A CclException object cannot be assigned and its constructors are intended for use by the CICS Transaction Gateway class implementation only.

Public methods

abendCode

```
const char* abendCode()
```

Returns a null-terminated string containing the ECI abend code, or blanks if no abend code is available.

className

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

Returns the name of the class in which the exception was raised.

diagnose

```
const char* diagnose() const
```

Returns text explaining the exception for use in diagnostic output, for example:

```
unknown server, classname=CclFlowI, methodName=afterReply, originCode=13  
"link", flowId=2, retCode=-22, abendCode="    "
```

exCode

```
Ccl::ExCode exCode() const
```

Returns the exception code. See Table 22 on page 233.

exCodeText

```
const char* exCodeText() const
```

Returns a text string that describes the exception code.

exObject

```
void* exObject() const
```

This method is relevant to to both the ECI and EPI.

exObject returns a pointer to the object controlling any server interaction at the time of the exception. If there was no such object, a null pointer is returned.

- In the case of ECI the pointer should be cast to a **CclFlow***. For example:
`CclFlow* pFlw = (CclFlow*) ex.exObject();`
- In the case of EPI **exObject** returns the relevant **CclTerminal** object pointer in the exception block. Cast this to a **CclTerminal***; for example:
`CclTerminal* pTerm = (CclFlow*)ex.exObject();`

methodName

const char* methodName() const

Returns the name of the method in which the exception was raised.

CclField class

An object of the **CclField** class is responsible for looking after a single field on a 3270 screen. **CclField** objects are created and deleted when 3270 data from the CICS server is processed by a **CclScreen** object.

Methods in this class allow field text and attributes to be read and updated. Modified fields are sent to the CICS server on the next **send**.

Public methods

appendText (1)

void appendText(const char* text, unsigned short length)

text

The text to be appended to the field

length

The number of characters to be appended to the field

Appends *length* characters from *text* to the end of the text already in the field.

appendText (2)

void appendText(const char* text)

text

The null-terminated text string to be appended to the field

Appends the characters within the *text* string to the end of the text already in the field.

backgroundColor

Color backgroundColor() const

Returns an enumeration indicating the background color of the field. The possible values are shown under **Color** at the end of the description of this class.

baseAttribute

char baseAttribute() const

Returns the 3270 base attribute of the field.

column

unsigned short column() const

Returns the column number of the position of the start of the field on the screen, with the leftmost column being 1.

dataTag

BaseMDT dataTag() const

Returns an enumeration indicating whether the data in the field has been modified. Possible values are:

- modified
- unmodified

foregroundColor

Color foregroundColor() const

Returns an enumeration indicating the foreground color of the field. The possible values are shown under **Color** at the end of the description of this class.

highlight

Highlight highlight() const

Returns an enumeration indicating which type of highlight is being used. The possible values are shown under **Highlight** at the end of the description of this class.

inputProt

BaseProt inputProt() const

Returns an enumeration indicating whether the field is protected. Possible values are:

- protect
- unprotect

inputType

BaseType inputType() const

Returns an enumeration indicating the input data type for this field. Possible values are:

- alphanumeric
- numeric

intensity

BaseInts intensity() const

Returns an enumeration indicating the field intensity. Possible values are :

- dark
- normal
- intense

length

unsigned short length() const

Returns the total length of the field. This includes one byte used to store the 3270 attribute byte information. The actual space for data is one byte less than the value returned by this method. See also the “textLength” on page 72 method.

position

unsigned short position() const

Returns the position of the start of the field on the screen, given by position = column number + ($n \times$ row number), where n is the number of columns in a row (usually 80).

resetDataTag

void resetDataTag()

Resets the modified data tag (MDT) to *unmodified*.

row

unsigned short row() const

Returns the row number of the position of the start of the field on the screen. The top row is 1.

setBaseAttribute

void setBaseAttribute(char attribute)

attribute

The value of the base 3270 attribute byte to be entered into the field

Sets the 3270 base attribute.

setExtAttribute

void setExtAttribute(char attribute, char value)

attribute

The type of extended attribute being set

value

The value of the extended attribute

Sets an extended 3270 attribute. If an invalid 3270 attribute type or value is supplied, a parameter exception is raised.

setText (1)

These methods update the field with the given text.

void setText(const char* text, unsigned short length)

text

The text to be entered into the field

length

The number of characters to be entered into the field

Copies *length* characters from *text* into the field.

setText (2)

void setText(const char* text)

text

The null-terminated text to be entered into the field

Copies *text*, without the terminating null, into the field.

text

const char* text() const

Returns the text currently held in the field.

textLength

unsigned short textLength() const

Returns the number of characters currently held in the field.

transparency

Transparency transparency() const

Returns an enumeration indicating the background transparency of the field. Possible values are shown under **Transparency** at the end of the description of this class.

Enumerations

BaseInts

Indicates the field intensity. Possible values are:

- normal
- intense
- dark

BaseMDT

Indicates whether data in the field has been modified. Possible values are:

- unmodified
- modified

BaseProt

Indicates whether the field is protected. Possible values are:

- protect
- unprotect

BaseType

Indicates field input data type. Possible values are:

- alphanumeric
- numeric

Color

Possible values are:

defaultColor	yellow	paleGreen
blue	neutral	paleCyan
red	black	gray
pink	darkBlue	white
green	orange	
cyan	purple	

Highlight

Indicates which type of highlight is being used. Possible values are:

defaultHlt	blinkHlt	underscoreHlt
normalHlt	reverseHlt	intenseHlt

Transparency

Indicates the background transparency of the field. Possible values are:

defaultTran
default transparency

orTran
OR with underlying color

xorTran
XOR with underlying color

opaqueTran
opaque

CclFlow class

A CclFlow object is used to control ECI communications for a client/server pair and to determine the synchronization of reply processing. Refer to *Compiling and Linking in the CICS Transaction Gateway: Programming Guide* for an explanation of synchronization. **CclFlow** automatically calls its **handleReply** method when a reply is available; this simplifies control of interleaved replies. Subclass **CclFlow** to implement your own **handleReply** method.

A CclFlow object is created for each client/server interaction (request from client and response from server). CclFlow objects can be reused when they become inactive, that is, when reply processing is complete. An attempt to delete or reuse an active CclFlow object raises an activeFlow exception.

CclFlow constructor

CclFlow (1)

CclFlow(Ccl::Sync syncType, unsigned long stackPages = 3)

syncType

The type of synchronization

stackPages

If asynchronous, the number of 4kb stack pages. The default is 3. If not asynchronous, this parameter is ignored.

CclFlow (2)

**CclFlow(Ccl::Sync syncType,
 unsigned long stackPages,
 const unsigned short &timeout)**

syncType

The type of synchronization

stackPages

If asynchronous, the number of 4kb stack pages. If not asynchronous, this parameter is ignored.

timeout

The time in seconds to wait for the ECI program to respond. If a timeout occurs, the HandleException method is called with a timeout CclException Object. Valid values are 0-32767.

Public methods

abendCode

const char* abendCode() const

Returns the abend code from the most recently executed CICS transaction, or blank if there have been none.

callType

CallType callType() const

Returns an enumeration value indicating the most recent type of server request.

callTypeText

const char* callTypeText() const

Returns the name of the most recent server request.

connection

CclConn* connection() const

Returns a pointer to the CclConn object that represents the server being used, if any, or zeros.

diagnose

const char* diagnose() const

Returns text explaining the exception for use in diagnostic output; for example:

```
"link", flowId=2, retCode=-22, abendCode="    "
```

flowId

unsigned short flowId() const

Returns the unique identity of this CclFlow object.

forceReset

void forceReset()

Makes the flow inactive and resets it. This is typically used to prepare a CclFlow object for re-use or deletion after a flow has been abandoned, for example when a C++ throw is used in an exception handler. This applies only to dsync and async flows. You cannot issue this on a sync call from another thread.

handleReply

virtual void handleReply(CclBuf* commarea)

commarea

A pointer to the CclBuf object containing the returned COMMAREA or zero if none.

This method is called whenever a reply is received from a server, irrespective of the type of synchronization or the type of call. See *Compiling and Linking* in the *CICS Transaction Gateway: Programming Guide*. To deal with replies, you should subclass **CclFlow** and provide your own implementation of **handleReply**. The default implementation merely returns to the caller.

listState

const char* listState() const

Returns a formatted string containing the current state of the object. For example:

```
Flow state..&CclFlow=000489A4 &CclFlowI=00203B70
syncType=2 threadId=0 stackPages=9 callType=0 flowId=0 commLength=0
retCode=0 systemRC=0 abendCode="  " &CclConnI=00000000 &CclUOWI=00000000
```

poll

Ccl::Bool poll(CclBuf* commarea = 0)

commarea

An optional pointer to the CclBuf object that will be used to contain the returned COMMAREA.

Returns an enumeration, defined within the **Ccl** class indicating whether a reply has been received from a deferred synchronous **Backout**, **Cancel**, **Changed**, **Commit**, **Link**, or **Status** call request. If **poll** is used on a flow object that is not deferred synchronous, a syncType exception is raised. Possible values are:

- yes** A reply has been received. **handleReply** has been called synchronously.
- no** No reply has been received. The client process is not blocked.

setTimeout

void setTimeout(const unsigned short &timeout)

timeout

the defined time in seconds to wait for the ECI program to respond. If a timeout occurs, the HandleException method is called with a timeout CclException Object. Valid values are 0-32767.

Sets the timeout value for the flow object for the next activation of the flow. This value can be set while a flow is active but does not affect the current active flow

syncType

Ccl::Sync syncType() const

Returns an enumeration, defined within the **Ccl** class indicating the type of synchronization being used. Possible values are shown in “Sync” on page 49.

timeout

short timeout()

Retrieves the current timeout value set for the flow object.

uow

CclUOW* uow() const

Returns a pointer to any CclUOW object containing information on any units of work (UOWs) associated with this interaction.

wait

void wait()

Waits for a reply from the server, blocking the client process in the meantime. If **wait** is used on a synchronous flow object, a syncType exception is raised.

Enumerations

CallType

The possible values for server requests in progress under the control of a CclFlow object are:

inactive

No server call is currently in progress

- link** A **CclConn::link** call to a server program
- backout**
A **CclUOW::backout** call to back out changes made to recoverable resources on the server
- commit**
A **CclUOW::commit** call to commit changes made to recoverable resources on the server
- status** A **CclConn::status** call to determine the status of a server connection
- changed**
A **CclConn::changed** call to request notification when the status of a connection to a server changes
- cancel**
A **CclConn::cancel** call to cancel an earlier **CclConn::changed** request.
-

CclMap class

The **CclMap** class is a base class for map classes created by the CICS BMS Map Conversion Utility. The methods provided by **CclMap** class are inherited by the classes generated from BMS maps.

CclMap constructor

CclMap(CclScreen* screen)

screen

A pointer to the matching **CclScreen** object.

Creates a **CclMap** object and checks (validates) that the map matches the content of the screen, defined by the **CclScreen** object. If validation was unsuccessful, an **invalidMap** exception is raised. If the supplied **CclScreen** object is invalid, a **parameter** exception is raised.

Public methods

exCode

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

Ccl::ExCode exCode() const

Returns an enumeration indicating the most recent exception code. The possible values are listed in Table 22 on page 233.

exCodeText

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

const char* exCodeText() const

Returns a text string describing the most recent exception code.

field (1)

CclField* field(unsigned short *index*)

index

The index number of the required CclField object.

Returns a pointer to the CclField object identified by *index* in the BMS map.

field (2)

CclField* field(unsigned short *row*, unsigned short *column*)

row

The row number of the required CclField object within the map. The top row is 1.

column

The column number of the required CclField object within the map. The left column is 1.

Returns a pointer to the CclField object identified by position in the BMS map.

Protected methods

namedField

CclField* namedField(unsigned long *index*)

index

The index number of the required CclField object.

Returns the address of the *index*th object.

validate

```
void validate(const MapData* map,
              const FieldIndex* index,
              const FieldData* fields)
```

map

A structure that contains information about the map. The structure is defined within this class and contains the following members, which are all unsigned short integers:

row Map row position on screen
col Map column position on screen
width Map width in columns
depth Map depth in rows
fields Number of fields
labels Number of labeled fields

index

The index number of the required CclField object. **FieldIndex** is a typedef of this class and is equivalent to an unsigned short integer.

fields

A structure that contains information about a particular field. The structure is defined within this class and contains the following members, which are all unsigned short integers:

row Field row (within map)
col Field column (within map)
len Field length

Validate map against the current screen.

CclScreen class

The **CclScreen** EPI class maintains all data on the 3270 virtual screen and provides access to this data. It contains a collection of CclField objects which represent the fields on the current 3270 screen.

A single CclScreen object is created by the CclTerminal object; use the **screen** method on the CclTerminal object to obtain it. The CclScreen object is updated by the CclTerminal object when 3270 data is received from CICS. A `dataStream` exception is raised if an unsupported data stream is received.

Public methods

cursorCol

unsigned short cursorCol() const

Returns the column number of the current position of the cursor. The left column is 1.

cursorRow

unsigned short cursorRow() const

Returns the row number of the current position of the cursor. The top row is 1.

depth

unsigned short depth() const

Returns the number of rows in the screen.

field (1)

These methods allow you to access fields on the current screen by returning a pointer to the relevant CclField object.

CclField* field(unsigned short *index*)

index

The index number of the field of interest

field (2)

CclField* field(unsigned short *row*, unsigned short *column*)

row

The row number of the field

column

The column number of the field

fieldCount

unsigned short fieldCount() const

Returns the number of fields in the screen.

mapName

const char* mapName()

Returns a padded null terminated string specifying the name of the map that was most recently referenced in the MAP option of a SEND MAP command processed for the terminal resource. If the terminal resource is not supported by BMS, or the server has no record of any map being sent, the value returned is spaces.

mapSetName

const char* mapSetName()

Returns a padded null terminated string specifying the name of the mapset that was most recently referenced in the MAPSET option of a SEND MAP command processed for the terminal resource. If the MAPSET option was not specified on the most recent request, BMS used the map name as the mapset name. In both cases, the mapset name used may have been suffixed by a terminal suffix. If the terminal resource is not supported by BMS, or the server has no record of any mapset being sent, the value returned is spaces.

setAID

void setAID(const AID key)

key

An AID key. See the “AID” on page 83 enumerations at the end of this chapter.

Sets the AID key value to be passed to the server on the next transmission.

setCursor

void setCursor(unsigned short row, unsigned short col)

row

The required row number of the cursor. The top row is 1.

col

The required column number of the cursor. The left column is 1.

Requests that the cursor position be set. If the supplied row or column values are outside the screen boundaries, a parameter exception is raised.

width

unsigned short width() const

Returns the number of columns on the screen.

Enumerations

AID

Indicates an AID key. Possible values are:

- enter
- clear
- PA1—PA3
- PF1—PF24

CclSecAttr

The CclSecAttr class provides information about passwords reported back by the external security manager when verifyPassword or changePassword methods are issued on CclConn or CclTerminal objects.

This object is created and owned by the CclConn or CclTerminal Object; access to this object is provided when the verifyPassword or changePassword methods are invoked.

Public Methods

expiryTime

CclSecTime* expiryTime() const

Returns a CclSecTime object that contains the Date and Time at which the password will expire

invalidCount

unsigned short invalidCount() const

Returns the Number of times that an invalid password has been entered for the userid.

lastAccessTime

CclSecTime* lastAccessTime() const

Returns a CclSecTime object that contains the date and time when the userid was last accessed.

lastVerifiedTime

CclSecTime* lastVerifiedTime() const

Returns a CclSecTime object that contains the date and time of the Last Verification.

CclSecTime

The CclSecTime class provides date and time information in the CclSecAttr object for various entries reported back by the external security manager when verifyPassword or changePassword methods are issued on CclConn or CclTerminal objects.

These objects are created and owned by the CclSecAttr object and access is obtained via the various methods available on this object. No Constructors or Destructors are available.

Public Methods

day

unsigned short day() const

Returns the day with a range from 1 to 31; 1 represents the first day of the month.

get_time_t

time_t get_time_t() const

Returns the date and time in a time_t format.

get_tm

tm get_tm() const

Returns the date and time in a tm structure.

hours

unsigned short hours() const

Returns the hours with a range from 0 to 23.

hundredths

unsigned short hundredths() const

Returns the hundredths of seconds with a range from 0 to 99.

minutes

unsigned short minutes() const

Returns the minutes with a range from 0 to 59.

month

unsigned short month() const

Returns the month with a range from 1 to 12. January is 1.

seconds

unsigned short seconds() const

Returns the seconds with a range from 0 to 59.

year

unsigned short year() const

Returns a 4-digit year

CclSession class

The **CclSession** class allows the programmer to implement reusable code to handle a segment (one or more transmissions) of a 3270 conversation. In multi-threaded environments it provides asynchronous handling of replies from CICS.

The **CclSession** class controls the flow of data to and from CICS within a single 3270 session. You should derive your own classes from **CclSession**.

CclSession constructor

CclSession(Ccl::Sync *syncType*)

syncType

The protocol to be used on transmissions to the CICS server. Possible values are:

async asynchronous
dsync deferred synchronous
sync synchronous

Public methods

diagnose

const char* diagnose() const

Returns a text description of the last error.

handleReply

virtual void handleReply(State state, CclScreen* screen)

state

An enumeration indicating the state of the data flow. The scope of the values is shown under **State** at the end of the description of this class.

screen

A pointer to the CclScreen object.

This is a virtual method which you can override when you develop your own class derived from **CclSession**. It is called when data is received from CICS.

state

State state() const

Returns an enumeration indicating the current state of the session. Possible values are shown under **State** at the end of the description of this class.

terminal

CclTerminal* terminal() const

Returns a pointer to the CclTerminal object for this session. This method returns a NULL pointer until the CclSession object has been associated with a CclTerminal object (that is, until the CclSession object has been used as a parameter on a CclTerminal **send** method).

transID

const char* transID() const

Returns the 4-letter name of the current transaction.

Enumerations

State

Indicates the state of a session. Possible values are:

idle The terminal is connected and no CICS transaction is in progress.

server A CICS transaction is in progress in the server.

client A CICS transaction is in progress, and the server is waiting for a response from the client.

discon The terminal is disconnected.

error There is an error in the terminal.

CclTerminal class

An object of class **CclTerminal** represents a 3270 terminal connection to a CICS server. A CICS connection is established when the object is created. Methods can then be used to converse with a 3270 terminal application (often a BMS application) in the CICS server.

The EPI must be initialized (that is, a CclEPI object created) before a CclTerminal object can be created.

The CclTerminal class destructor does not purge ATI requests queued against the terminal.

CclTerminal constructor

CclTerminal (1)

```
CclTerminal(const char* server = NULL,  
            const char* devtype = NULL,  
            const char* netname = NULL)
```

server

The name of the server with which you want to communicate. If no name is provided the default server system is assumed. The length is adjusted to 8 characters by padding with blanks.

devtype

The name of the model terminal definition that the server uses to generate a terminal resource definition. If no string is provided the default model is used. The length is adjusted to 16 characters by padding with blanks.

netname

The name of the terminal resource to be installed or reserved. The default is to use the contents of *devtype*. The length is adjusted to 8 characters by padding with blanks.

Creates the CclTerminal object that is used for EPI communication between the client and server.

This constructor does an implicit install terminal. You do not need to invoke the install method if you construct a terminal object this way.

If the named server is not configured in the CICS Transaction Gateway initialization file, an unknownServer exception is raised.

If invalid values are supplied for *server*, *devtype* or *netname*, a parameter exception is raised.

If a CclEPI object has not been created, an initEPI exception is raised.

If the maximum number of supported terminal connections has been exceeded, a maxRequests exception is raised.

CclTerminal (2)

```
CclTerminal(const char* server,  
             const char* devtype,  
             const char* netname,  
             signonType signonCapability  
             const char* userid  
             const char* password  
             const unsigned short &readTimeOut,  
             const unsigned short &CCSid)
```

server

The name of the server with which you want to communicate. If no name is provided the default server system is assumed. The length is adjusted to 8 characters by padding with blanks.

devtype

The name of the model terminal definition which the server uses to generate a terminal resource definition. If no string is provided the default model is used. The length is adjusted to 16 characters by padding with blanks.

netname

The name of the terminal resource to be installed or reserved. The default is to use the contents of *devtype*. The length is adjusted to 8 characters by padding with blanks.

signonCapability

Sets the type of sign-on capability for the terminal.

Possible values are:

- `Cc1Terminal::SignonCapable`
- `Cc1Terminal::SignonIncapable`

userid

The name of the userid to associate with this terminal resource

password

The password to associate with the userid

readTimeOut

A value in the range 0 through 3600, specifying the maximum time in seconds between the time the classes go clientrepl state and the application program invokes the reply method.

CCSid

An unsigned short specifying the coded character set identifier (CCSID) that identifies the coded graphic character set used by the Client application for data passed between the terminal resource and CICS transactions. A zero string means that a default will be used.

Creates a Terminal object that does not do an implicit install terminal. You must run the install method to install the terminal.

Public methods

alterSecurity

void alterSecurity(const char* *userid*,const char* *password*)

userid

The new userid

password

The new password for *userid*

Allows you to re-define the userid and password for a terminal resource. You may call the method before you install a terminal. It changes only the terminal definition; the new userid and password will be used for the terminal when install is called.

changePassword

Cc1SecAttr* changePassword(const char* *newPassword*)

newPassword

The new password

Allows a Client application to change the password held in the terminal object and the password recorded by an external security manager for the userid held in the terminal object. The external security manager is assumed to be located in the server defined by the terminal object.

CCSid

unsigned short CCSid()

Returns the selected code page as an unsigned short.

diagnose

const char* diagnose()

Returns a character string that holds a description of the error returned by the most recent server call.

disconnect (1)

void disconnect()

Disconnects the terminal from CICS. No attempt is made to purge any outstanding running transaction.

disconnect (2)

void disconnect(Ccl::Bool *withPurge*)

withPurge

Ccl::Yes

Disconnects the terminal from CICS and attempts to purge any outstanding running transaction. This purge function does not cancel ATI requests queued against the terminal.

Ccl::No

Disconnects the terminal from CICS. No attempt is made to purge outstanding running transactions.

discReason

void discReason(void)

Returns the reasons for a disconnection. See “EndTerminalReason” on page 96.

exCode

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

Ccl::ExCode exCode() const

Returns an enumeration indicating the most recent exception code. The possible values are listed in Table 22 on page 233.

exCodeText

Deprecated method

: Do not use this method in new applications. The method has been deprecated and is provided only for compatibility.

const char* exCodeText() const

Returns a text string describing the most recent exception code.

install

```
void install(CclSession *session,  
             const unsigned short &installTimeOut)
```

session

A pointer to the CclSession object that is to be used for the CICS server interaction.

installTimeOut

A value in the range 0 to 3600, specifying the maximum time in seconds that installation of the terminal resource is allowed to take. A value of 0 means that no limit is set.

Connects a non-connected terminal resource. Throws an invalidState error if already connected, or a timeout error if a timeout occurs.

makeSecurityDefault

```
void makeSecurityDefault()
```

Informs the client that the current userid and password for this object are to become the default for ECI and EPI requests passed to the server as specified in the construction of the Terminal object.

netName

const char* netName() const

Returns the network name of the terminal as a null terminated string.

password

const char* password()

Returns a null terminated string containing the current password setting for the terminal, or null if the terminal has no password.

poll

Ccl::Bool poll()

Polls for data from the CICS server.

For deferred synchronous transmissions (that is, if a deferred synchronous CclSession object was used on a previous send call) the **poll** method is called by the application when it wants to receive data from the CICS server. If a reply from CICS is ready, the CclTerminal object updates the CclScreen object with the contents of the 3270 data stream received from CICS, the **handleReply** virtual function on the CclSession object is called, and the **poll** method returns Ccl::yes. If no reply has been received from CICS, the **poll** method returns Ccl::no.

The **poll** method is used only for deferred synchronous transmissions; a syncType exception is raised if the poll method is called when a synchronous or asynchronous session is in use. An invalidState exception is raised if the **poll** method is called when there was no previous **send** call. The CclTerminal object should be in server state for poll to be called.

A CICS server transaction may send more than one reply in response to a CclTerminal **send** call. More than one CclTerminal **poll** call may therefore be needed to collect all the replies. Use the CclTerminal **state** method to find out if further replies are expected. If there are, the value returned will be server. See *EPI call synchronization types* in the *CICS Transaction Gateway: Programming Guide*.

queryATI

ATIState queryATI()

Returns an enumeration indicating whether the “Automatic Transaction Initiation” (ATI) is enabled or disabled. Possible values are:

- disabled
- enabled

readTimeout

const char* readTimeout()

Returns the read timeout value for the terminal as a null terminated string .

receiveATI

void receiveATI(CclSession* session)

session

pointer to the CclSession object that is to be used for the CICS server interaction.

Waits for and receives 3270 data stream for a CICS ATI transaction. The CclSession object supplied as a parameter determines whether the call is synchronous or asynchronous, and can be subclassed to provide a reply handler

screen

CclScreen* screen() const

Returns a pointer to the CclScreen object that is handling the 3270 screen associated with this terminal session.

send (1)

**void send(CclSession* session,
 const char* transid,
 const char* startdata = NULL)**

session

A pointer to the CclSession object that controls the session which is to be used. If no valid CclSession object is supplied, a parameter exception is raised.

transid

The name of the transaction which is to be started

startdata

start transaction data. The default is to have no data for the transaction being started.

Formats and sends a 3270 data stream, starting the named transaction. The CclTerminal object must be in idle state (connected to a CICS server but with no transaction in progress). If the object is not in idle state, an invalidState exception is raised.

send (2)

void send(CclSession* session)

The *session* parameter is described above.

Formats and sends a 3270 data stream. The CclTerminal object must be in idle state (see above) or in client state (that is, with a transaction in progress and the CICS server waiting for a response). If the object is not in idle or client state, an invalidState exception is raised.

setATI

void setATI(ATIState newstate)

newstate

An enumeration indicating whether the ATI is to be enabled or disabled. The scope of the values is within this class and the possible values are disabled and enabled.

signonCapability

signonType signonCapability()

Returns the type of sign-on capability applied to the terminal at installation.

Possible values are:

- CclTerminal::signonCapable
- CclTerminal::signonIncapable
- CclTerminal::signonUnknown

state

State state() const

Returns an enumeration indicating the current state of the session. Possible values are shown at the end of the description of this class.

serverName

const char* serverName() const

Returns the name of the CICS server to which this terminal session is connected.

termID

const char* termID() const

Returns the 4-character terminal ID.

transID

const char* transID() const

Returns the 4-character name of the current CICS transaction. If a RETURN IMMEDIATE is run from the current transaction, TransId does not provide the name of the new transaction; it still contains the name of the first transaction.

userid

const char* userId()

Returns a null terminated string containing the current userid setting for the terminal, Null if none.

verifyPassword

CclSecAttr* verifyPassword()

Allows a Client application to verify that the password held in the terminal object matches the password recorded by an external security manager for the userid held in the terminal object. The external security manager is assumed to be located in the server defined by the terminal object.

Enumerations

ATIState

Indicates whether “Automatic Transaction Initiation” (ATI) is enabled or disabled. Possible values are:

- enabled
- disabled

signonType

Indicates the sign-on capability of a terminal. Possible values are:

signonCapable

Sign-on Capable

signonIncapable

Sign-on Incapable

signonUnknown

Sign-on Unknown

State

Indicates the state of the CclTerminal object. Possible values are:

client A CICS transaction is in progress and the server is waiting for a response from the client.

discon

The terminal is disconnected.

error There is an error in the terminal.

idle The terminal is connected and no CICS transaction is in progress.

server

A CICS transaction is in progress in the server.

termDefined

A terminal has been defined but not installed.

txnTimedOut

A conversational transaction has timed out, but the END_TRAN event has not been retrieved. For synchronous and asynchronous terminals the terminal method blocks until the event has been received and the terminal becomes idle. For deferred synchronous terminals it indicates that a poll() needs to be done to get the event. This resets the terminal to the idle state; handleException() and handleReply() are not invoked.

EndTerminalReason

Indicates the EndTerminalReason of the CclTerminal object. Possible values are:

signoff

A disconnect was requested or the user has signed off the terminal.

shutdown

The CICS server has been shutdown.

outofService

The terminal has been switched to out of service.

unknown

An unknown situation has occurred.

failed The terminal failed to disconnect.

notDiscon

The terminal is not disconnected.

CclUOW class

Use this ECI class when you make updates to recoverable resources in the server within a “unit of work” (UOW). Each update in a UOW is identified at the client by a reference to its **CclUOW**—see **link** in **CclConn** (“link” on page 58).

A CclUOW object cannot be copied or assigned. An attempt to delete a CclUOW object for which there is an active CclFlow object raises an activeFlow exception. Any attempt to delete an active CclUOW object, that is one which has not been committed or backed out, raises an activeUOW exception.

CclUOW constructor

CclUOW()

Creates a CclUOW object.

Public methods

backout

void backout(CclFlow& *flow*)

flow

A reference to the CclFlow object that is used to control the client/server call

Terminate this UOW and back out all changes made to recoverable resources in the server.

commit

void commit(CclFlow& *flow*)

flow

A reference to the CclFlow object that is used to control the client/server call

Terminate this UOW and commit all changes made to recoverable resources in the server.

forceReset

void forceReset()

Make this UOW inactive and reset it.

listState

const char* listState() const

Returns a zero-terminated formatted string containing the current state of the object. For example:

```
UOW state..&Cc1UOW=0004899C &Cc1UOWI=00203BD0  
&Cc1ConnI=00000000 uowId=0 &Cc1FlowI=00000000
```

uowId

unsigned long uowId() const

Returns the identifier of the UOW. 0 means that the UOW is either complete or has not yet started. In other words, it is inactive.

Chapter 4. C and COBOL

COBOL is only supported on AIX and Windows platforms.

The callback functions of ECI and EPI are not supported in COBOL applications.

External Call Interface

CICS_ExternalCall

ECI_Parms

Purpose

CICS_ExternalCall gives access to the program link calls, status information calls, and reply solicitation calls. The function performed is controlled by the **eci_call_type** field in the ECI parameter block.

Parameters

ECI_Parms

A pointer to the ECI parameter block. *Set the parameter block to nulls before use.* The parameter block fields that are used as input and output are described in detail for each call type in the following sections. A brief summary of the fields follows:

eci_call_type

An integer field defining the type of call being made. For details of the functions provided, see *Types of ECI call* in *CICS Transaction Gateway: Programming Guide*.

eci_program_name

The name of a program to be called.

eci_userid

User ID for security checking.

eci_password

Password for security checking.

eci_transid

A transaction identifier.

eci_abend_code

Abend code for a failed program.

eci_commarea

A COMMAREA for use by a called program, or for returned status information.

eci_commarea_length

The length of the COMMAREA. The size of the COMMAREA must be set to the largest size of the input or output data. This length must not exceed 32 500 bytes. If the input data is less than the length of the COMMAREA, pad the COMMAREA with nulls. The Client daemon strips off the null padding and sends only the data on the ECI request to the CICS server.

eci_timeout

The time to wait for a response from the CICS server. For more information on the ECI time-out support, see the *CICS Transaction Gateway: Programming Guide*.

reserved1

A return code giving more information about an unexpected error.

This field was previously **eci_system_return_code**. In Version 3.1 and higher of the product, this field is kept for compatibility. No information is returned in this field; all system errors are written to the CICS Transaction Gateway's error log.

eci_extend_mode

Used to manage logical units of work that span multiple ECI requests. See *CICS Transaction Gateway: Programming Guide* for more details.

eci_message_qualifier

A user-provided reference to an asynchronous call.

eci_luw_token

An identifier for a logical unit of work.

eci_sysid

Reserved for future use; leave null.

eci_version

The version of the ECI for which the application is coded. Use the value ECI_VERSION_1A.

eci_system_name

The name of a CICS server.

eci_callback

A pointer to a callback routine for an asynchronous request. Not supported in COBOL applications.

eci_userid2

User ID for security checking. This is used if the User ID or password is more than 8 Characters.

eci_password2

Password for security checking. This is used if the User ID or password is more than 8 Characters.

eci_tpn

A transaction identifier for the mirror transaction.

Return Codes

In addition to the return codes described for each call type in the following sections, the following return codes are possible.

ECI_ERR_INVALID_CALL_TYPE

The call type was not one of the valid call types.

ECI_ERR_CALL_FROM_CALLBACK

The call was made from a callback routine.

ECI_ERR_REQUEST_TIMEOUT

The time-out interval expired before the request could be processed, or the specified interval was negative.

ECI_ERR_RESPONSE_TIMEOUT

The time-out interval expired while the program was running.

ECI_ERR_SYSTEM_ERROR

An internal system error occurred. The error might have been in the CICS Transaction Gateway or in the server. The programmer should save the information returned in the CICS Transaction Gateway's error log, as this will help service personnel to diagnose the error.

ECI_ERR_INVALID_VERSION

The value supplied for **eci_version** was invalid.

In some implementations, some of the return codes documented here and for each call type will never be returned.

The mapping of actual return code values to the symbolic names is contained in the following file for the Windows operating systems:

C <install_path>\include\cics_eci.h

Cobol <install_path>\copybook\cicseci.cbl

and in the following files for the UNIX and Linux operating systems:

C <install_path>/include/cics_eci.h

Call types for the CICS_ExternalCall

ECI_SYNC call type

Environment

The ECI_SYNC call type is available in all environments.

Purpose

The ECI_SYNC call type provides a synchronous program link call to start, continue, or end a logical unit of work. The calling application does not get control back until the called CICS program has run to completion.

ECI parameter block fields

The ECI parameter block should be set to nulls before setting the input parameter fields.

eci_call_type

Required input parameter, which must be set to ECI_SYNC.

eci_program_name

Input parameter, required except when **eci_extend_mode** is ECI_COMMIT or ECI_BACKOUT. (See the *Logical units of work in ECI* table in *CICS Transaction Gateway: Programming Guide* for more details.)

An 8-character field containing the name of the program to be called. Pad unused characters with spaces. This field is transmitted to the server without conversion to uppercase.

The characters used are translated from the client's code page to an EBCDIC code page before transmission. If the server uses an ASCII code page, they will be retranslated. The only characters guaranteed to be invariant under these translations are the uppercase characters A to Z, and the numeric characters 0 to 9. Some EBCDIC servers (Katakana and Hebrew character set A) do not use the standard representations of the lowercase alphabetic characters; use them with care when communicating with such servers.

eci_userid

Required input parameter.

An 8-character field containing a user ID. Pad unused characters with spaces.

Consult the documentation for the CICS Transaction Gateway and the server to check whether this field is converted to upper case before being transmitted to the server. If a user ID or password longer than 8

characters is required, set **eci_userid** and **eci_password** to nulls, and use fields **eci_userid2** and **eci_password2** instead.

If a user ID is supplied, the server uses the user ID and any supplied password to authenticate the user. The supplied user ID and password are used in subsequent security checking in the server.

eci_password

Required input parameter.

An 8-character field containing a password. Pad unused characters with spaces.

Consult the documentation for the CICS Transaction Gateway and the server to check whether this field is converted to upper case before being transmitted to the server. If a user ID or password longer than 8 characters is required, set this field and **eci_userid** to nulls, and use fields **eci_userid2** and **eci_password2** instead.

eci_transid

Optional input parameter

A 4-character field optionally containing the ID of a CICS transaction. Pad unused characters with spaces. The parameter is ignored if **eci_tpn** is used (set to any value other than nulls). The use of this parameter depends on the client from which the request is sent. The value of **eci_transid** is converted from ASCII to EBCDIC, with no upper case translation, and stored in EIBTRNID for the duration of the LINK to the program specified in the **eci_program_name**.

The called program runs under the mirror transaction CPMI, but is linked to under the **eci_transid** transaction name. This name is available to the called program for querying the transaction ID. Some servers use the transaction ID to determine security and performance attributes for the called program. In those servers, use this parameter to control the processing of your called programs.

If the ECI request is extended (see the description of **eci_extend_mode**), the **eci_transid** parameter has a meaning only for the first call in the unit of work.

If the field is all nulls, and **eci_tpn** is not specified, the default server transaction ID is used.

eci_abend_code

Output parameter.

A 4-character field in which a CICS abend code is returned if the transaction that executes the called program abends. Unused characters are padded with spaces.

eci_commarea

Optional input parameter.

A pointer to the data to be passed to the called CICS program as its COMMAREA. The COMMAREA will be used by the called program to return information to the application.

If no COMMAREA is required, supply a null pointer and set the length (specified in **eci_commarea_length**) to zero.

If the code page of the application is different from the code page of the server, data conversion must be performed at the server. To do this, use CICS-supplied resource conversion capabilities, such as the DFHCNV macro definitions.

eci_commarea_length

Optional input parameter.

The length of the COMMAREA in bytes. This value may not exceed 32 500. (Some client/server combinations may allow larger COMMAREAs, but this is not guaranteed to work.)

If no COMMAREA is required, set this field to zero and supply a null pointer in **eci_commarea**.

eci_timeout

The time in seconds to wait for a response from the CICS server. A value of 0 means that no limit is set.

If timeout occurs, the conversation is abended.

reserved1

Output parameter.

This field was previously **eci_system_return_code**. In the CICS Transaction Gateway Version 3.1, and higher, this field is reserved for compatibility with earlier versions. No information is returned in this field; all system errors are written to the CICS Transaction Gateway's error log.

eci_extend_mode

Required input parameter.

An integer field determining whether a logical unit of work is terminated at the end of this call.) (See the *Logical units of work in ECI* table in *CICS Transaction Gateway: Programming Guide* for more details.)

The values for this field (shown by their symbolic names) are as follows:

ECI_NO_EXTEND

1. If the input **eci_luw_token** field is zero, this is the only call for a logical unit of work.

2. If the input **eci_luw_token** field is not zero, this is the last call for the specified logical unit of work.

In either case, changes to recoverable resources are committed by a CICS end-of-task syncpoint, and the logical unit of work ends.

If you set `eci_extend_mode` to `ECI_NO_EXTEND` and `eci_luw_token` to 0, you will observe one request flowing from client to server and one reply flowing from server to client. The server sends the reply after the program specified in `eci_program_name` has been invoked and the changes made by that program have been committed.

ECI_EXTENDED

1. If the input **eci_luw_token** field is zero, this is the first call for a logical unit of work that is to be continued.
2. If the input **eci_luw_token** field is not zero, this call is intended to continue the specified logical unit of work.

In either case the logical unit of work continues after the called program completes successfully, and changes to recoverable resources remain uncommitted.

ECI_COMMIT

Terminate the current logical unit of work, identified by the input **eci_luw_token** field, and commit all changes made to recoverable resources.

ECI_BACKOUT

Terminate the logical unit of work identified by the input **eci_luw_token** field, and back out all changes made to recoverable resources.

eci_luw_token

Required input and output parameter.

An integer field used for identifying the logical unit of work to which a call belongs. It must be set to zero at the start of a logical unit of work (regardless of whether the logical unit of work is going to be extended). If the logical unit of work is to be extended, the ECI updates **eci_luw_token** with a valid value on the first call of the logical unit of work, and this value should be used as input to all later calls related to the same logical unit of work. (See the *Logical units of work in ECI* table in *CICS Transaction Gateway: Programming Guide* for more details.)

If the return code is not `ECI_NO_ERROR`, and the call was continuing or ending an existing logical unit of work, this field is used as output to report the condition of the logical unit of work. If it is set to zero,

the logical unit of work has ended, and its updates have been backed out. If it is nonzero, it is the same as the input value, the logical unit of work is continuing, and its updates are still pending.

eci_sysid

Required input parameter.

Reserved for future use, but this field should be initialized with nulls before the start of each logical unit of work.

eci_version

Required input parameter.

The version of the ECI for which the application is coded. Use the value `ECI_VERSION_1A`.

eci_system_name

Optional input parameter.

An 8-character field that specifies the name of the server to which the ECI request is to be directed. Pad unused characters with spaces. If supplied, it should be one of the server names returned by **CICS_EciListSystems**. The value may be supplied whenever **eci_luw_token** is set to zero. (If it is supplied when **eci_luw_token** is not zero, it is ignored, because the server was established at the start of the logical unit of work.)

If the field is set to nulls, the default server is selected; the name of the chosen server is returned in this field, and must be used in subsequent related ECI requests. If ECI requests made in different logical units of work must be directed to the same server, **eci_system_name** must identify that server by name.

eci_userid2

Optional input parameter.

If the **eci_userid** field is set to nulls, the **eci_userid2** field specifies the user ID (if any) to be used at the server for any authority validation. The user ID can be up to 16 characters.

See the description of the **eci_userid** field for information about how the user ID is used.

eci_password2

Optional input parameter.

If the **eci_password** field is set to nulls, the **eci_password2** field specifies the password (if any) to be used at the server for any authority validation. The password can be up to 16 characters.

See the description of the **eci_password** field for information about how the password is used.

eci_tpn

Optional input parameter.

A 4-character field that specifies the transaction ID of the transaction that will be used in the server to process the ECI request. This transaction must be defined in the server as a CICS mirror transaction. If the field is not set, the default mirror transaction CPMI is used.

If the ECI request is extended (see the description of **eci_extend_mode**), this parameter has a meaning only for the first request.

If this field is used, the contents of **eci_transid** are ignored.

Return codes

See also the general list of return codes for **CICS_ExternalCall** in “CICS_ExternalCall ECI_Parms” on page 99.

ECI_NO_ERROR

The call completed successfully.

ECI_ERR_INVALID_DATA_LENGTH

The value in **eci_commarea_length** field is outside the valid range, or is inconsistent with the value in **eci_commarea**, being zero for a non-null **eci_commarea** pointer, or non-zero for a null **eci_commarea** pointer.

ECI_ERR_INVALID_EXTEND_MODE

The value in **eci_extend_mode** field is not valid.

ECI_ERR_NO_CICS

The CICS Transaction Gateway is unavailable, or the server implementation is unavailable, or a logical unit of work was to be begun, but the CICS server specified in **eci_system_name** is not available. No resources have been updated.

ECI_ERR_CICS_DIED

A logical unit of work was to be begun or continued, but the CICS server was no longer available. If **eci_extend_mode** was **ECI_EXTENDED**, the changes are backed out, and the logical unit of work ends. If **eci_extend_mode** was **ECI_NO_EXTEND**, **ECI_COMMIT**, or **ECI_BACKOUT**, the application cannot determine whether the changes have been committed or backed out, and must log this condition to aid future manual recovery.

ECI_ERR_TRANSACTION_ABEND

The CICS transaction that executed the requested program abended. The abend code will be found in **eci_abend_code**. For information

about abend codes and their meaning, consult the documentation for the server system to which the request was directed.

ECI_ERR_LUW_TOKEN

The value supplied in **eci_luw_token** is invalid.

ECI_ERR_ALREADY_ACTIVE

An attempt was made to continue an existing logical unit of work, but there was an outstanding asynchronous call for the same logical unit of work.

ECI_ERR_RESOURCE_SHORTAGE

The server implementation or the Client daemon did not have enough resources to complete the request.

ECI_ERR_NO_SESSIONS

A new logical unit of work was being created, but the application already has as many outstanding logical units of work as the configuration will support.

ECI_ERR_INVALID_DATA_AREA

Either the pointer to the ECI parameter block is invalid, or the pointer supplied in **eci_commarea** is invalid.

ECI_ERR_ROLLEDBACK

An attempt was made to commit a logical unit of work, but the server was unable to commit the changes, and backed them out instead.

ECI_ERR_UNKNOWN_SERVER

The requested server could not be located. Only servers returned by **CICS_EciListSystems** are acceptable.

ECI_ERR_MAX_SESSIONS

This is returned if the MAXREQUESTS limit, as defined in your configuration file, was exceeded.

ECI_ERR_MAX_SYSTEMS

You tried to start requests to more servers than your configuration allows. Consult the documentation for your CICS Transaction Gateway or server to see how to control the number of servers you can use.

ECI_ERR_SECURITY_ERROR

You did not supply a valid combination of user ID and password.

ECI_ASYNC call type

Environment

Purpose

The ECI_ASYNC call type provides an asynchronous program link call to start, continue, or end a logical unit of work. The calling application gets control back when the ECI has accepted the request. At this point the parameters have been validated; however, the request might still be queued for later processing.

If no callback routine is provided, the application must use a reply solicitation call to determine whether the request has ended and what the outcome was.

If a callback routine is provided, the callback routine **eci_callback** is invoked when a response is available.

Note: Some compilers do not support the use of callback routines. Consult your compiler documentation for more information.

It is important that the Eci parameter blocks of outstanding ECI_ASYNC calls are not modified before the results of the call are received. Results will be incorrect if these blocks are modified before this stage.

When the callback routine is called, it is passed a single parameter—the value specified in **eci_message_qualifier**. This enables the callback routine to identify the asynchronous call that is completing. Follow these guidelines when using the callback routine:

1. The minimum possible processing should be performed within the callback routine.
2. ECI functions cannot be invoked from within the callback routine.
3. The callback routine should indicate to the main body of the application that the reply is available using an appropriate technique for the operating system upon which the ECI application is executing. For example, in a multithreaded environment, the callback routine might post a semaphore to signal another thread that an event has occurred.
4. The application, not the callback routine, must use a reply solicitation call to receive the actual response.

ECI parameter block fields

Set the ECI parameter block to nulls before you set the input parameter fields.

eci_call_type

Required input parameter.

Must be set to ECI_ASYNC.

eci_program_name

Input only, required parameter except when **eci_extend_mode** is ECI_COMMIT or ECI_BACKOUT. (See the *Logical units of work in ECI* table in *CICS Transaction Gateway: Programming Guide* for more details.)

An 8-character field containing the name of the program to be called. Pad unused characters with spaces. This field is transmitted to the server without conversion to uppercase.

The characters used are translated from the client's code page to an EBCDIC code page before transmission. If the server uses an ASCII code page, they will be retranslated. The only characters guaranteed to be invariant under these translations are the uppercase characters A to Z, and the numeric characters 0 to 9. Some EBCDIC servers (Katakana and Hebrew character set A) do not use the standard representations of the lowercase alphabetic characters; use them with care when communicating with such servers.

eci_userid

Required input parameter.

An 8-character field containing a user ID. Pad unused characters with spaces.

Consult the documentation for the CICS Transaction Gateway and the server to check whether this field is converted to upper case before being transmitted to the server. (If a user ID or password longer than 8 characters is required, set **eci_userid** and **eci_password** to nulls, and use **eci_userid2** and **eci_password2** instead.)

If a user ID is supplied, the server uses the user ID and any supplied password to authenticate the user. The supplied user ID and password are used in subsequent security checking in the server.

eci_password

Required input parameter.

An 8-character field containing a password. Pad unused characters with spaces.

Consult the documentation for the CICS Transaction Gateway and the server to check whether this field is converted to upper case before being transmitted to the server. (If a user ID or password longer than 8 characters is required, set **eci_userid** and **eci_password** to nulls, and use **eci_userid2** and **eci_password2** instead.)

eci_transid

Optional input parameter

A 4-character field optionally containing the ID of a CICS transaction. Pad unused characters with spaces. The parameter is ignored if

eci_tpn is used (set to any value other than nulls). The use of this parameter depends on the client from which the request is sent. The value of **eci_transid** is converted from ASCII to EBCDIC, with no upper case translation, and stored in EIBTRNID for the duration of the LINK to the program specified in the **eci_program_name**.

The called program runs under the mirror transaction CPMI, but is linked to under the **eci_transid** transaction name. This name is available to the called program for querying the transaction ID. Some servers use the transaction ID to determine security and performance attributes for the called program. In those servers, use this parameter to control the processing of your called programs.

If the ECI request is extended (see **eci_extend_mode**), the **eci_transid** parameter has a meaning only for the first call in the unit of work.

If the field is all nulls, and **eci_tpn** is not specified, the default server transaction ID is used.

eci_commarea

Required input parameter.

A pointer to the data to be passed to the called CICS program as its COMMAREA.

If no COMMAREA is required, supply a null pointer and set the length (specified in **eci_commarea_length**) to zero.

If the code page of the application is different from the code page of the server, data conversion must be performed at the server. To do this, use CICS-supplied resource conversion capabilities, such as the DFHCNV macro definitions.

eci_commarea_length

Required input parameter.

The length of the COMMAREA in bytes. This value may not exceed 32 500. (Some client/server combinations may allow larger COMMAREAs, but this is not guaranteed to work.)

If no COMMAREA is required, set this field to zero and supply a null pointer in **eci_commarea**.

eci_timeout

The time in seconds to wait for a response from the CICS server. A value of 0 means that no limit is set.

If timeout occurs, the conversation is abended.

reserved1

Output parameter.

This field was previously **eci_system_return_code**. In the CICS Transaction Gateway Version 3.1, and higher, this field is reserved for compatibility with earlier versions. No information is returned in this field; all system errors are written to the error log.

eci_extend_mode

Required input parameter.

An integer field determining whether a logical unit of work is terminated at the end of this call. (See the *Logical units of work in ECI* table in *CICS Transaction Gateway: Programming Guide* for more details.)

Values (shown by their symbolic names) for this field are as follows:

ECI_NO_EXTEND

1. If the input **eci_luw_token** field is zero, this is the only call for a logical unit of work.
2. If the input **eci_luw_token** field is not zero, this is the last call for the specified logical unit of work.

In either case, changes to recoverable resources are committed by a CICS end-of-task syncpoint, and the logical unit of work ends.

ECI_EXTENDED

1. If the input **eci_luw_token** field is zero, this is the first call for a logical unit of work that is to be continued.
2. If the input **eci_luw_token** field is not zero, this call is intended to continue the specified logical unit of work.

In either case the logical unit of work continues after the called program completes, and changes to recoverable resources remain uncommitted.

ECI_COMMIT

Terminate the current logical unit of work, identified by the input **eci_luw_token** field, and commit all changes made to recoverable resources.

ECI_BACKOUT

Terminate the logical unit of work identified by the input **eci_luw_token** field, and back out all changes made to recoverable resources.

eci_message_qualifier

Optional input parameter.

An integer field allowing the application to identify each asynchronous call if it is making more than one. If a callback routine is specified, the value in this field is returned to the callback routine during the notification process.

eci_luw_token

Required input and output parameter.

An integer field used for identifying the logical unit of work to which a call belongs. It must be set to zero at the start of a logical unit of work (regardless of whether the logical unit of work is going to be extended), and the ECI updates it with a valid value on the first or only call of the logical unit of work. If the logical unit of work is to be extended, this value should be used as input to all later calls related to the same logical unit of work. (See the *Logical units of work in ECI* table in *CICS Transaction Gateway: Programming Guide* for more details.)

If the return code is not ECI_NO_ERROR, and the call was continuing or ending an existing logical unit of work, this field is used as output to report the condition of the logical unit of work. If it is set to zero, the logical unit of work has ended, and its updates have been backed out. If it is nonzero, it is the same as the input value, the logical unit of work is continuing, and its updates are still pending.

eci_sysid

Required input parameter.

Reserved for future use, but this field should be initialized with nulls before the start of each logical unit of work.

eci_version

Required input parameter.

The version of the ECI for which the application is coded. Use the value ECI_VERSION_1A.

eci_system_name

Optional input parameter.

An 8-character field that specifies the name of the server to which the ECI request is to be directed. Pad unused characters with spaces. The value may be supplied whenever **eci_luw_token** is set to zero. (If it is supplied when **eci_luw_token** is not zero, it is ignored, because the server was established at the start of the logical unit of work.)

If the field is set to nulls, the default server is selected. You can obtain the name of the chosen server from the **eci_system_name** field of the reply solicitation call you use to get the result of this asynchronous request. (If later ECI requests made in different logical units of work must be directed to the same server as this request, **eci_system_name** in those requests must identify that server by name.)

eci_callback

Optional input parameter.

A pointer to the routine to be called when the asynchronous request completes. (The callback routine will be called only if the return code is `ECI_NO_ERROR`, and the pointer is not null.)

eci_userid2

Optional input parameter.

If the `eci_userid` field is set to nulls, the `eci_userid2` field specifies the user ID (if any) to be used at the server for any authority validation. The user ID can be up to 16 characters.

See the description of the `eci_userid` field for information about how the user ID is used.

eci_password2

Optional input parameter.

If the `eci_password` field is set to nulls, the `eci_password2` field specifies the password (if any) to be used at the server for any authority validation. The password can be up to 16 characters.

See the description of the `eci_password` field for information about how the password is used.

eci_tpn

Optional input parameter.

A 4-character field that specifies the transaction ID of the transaction that will be used in the server to process the ECI request. This transaction must be defined in the server as a CICS mirror transaction. If the field is not set, the default mirror transaction CPMT is used.

If the ECI request is extended (see the description of `eci_extend_mode`), this parameter has a meaning only for the first request.

If this field is used, the contents of `eci_transid` are ignored.

Return codes

See also the general list of return codes for `CICS_ExternalCall` in “CICS_ExternalCall ECI_Parms” on page 99.

If the return code is not `ECI_NO_ERROR`, the callback routine will not be called, and there will be no asynchronous reply for this request.

ECI_NO_ERROR

The call to the ECI completed successfully. No errors have yet been detected. The callback routine will be called when the request completes.

ECI_ERR_INVALID_DATA_LENGTH

The value in `eci_commarea_length` field is outside the valid range, or is inconsistent with the value in `eci_commarea`, being zero for a non-null `eci_commarea` pointer, or non-zero for a null `eci_commarea` pointer.

ECI_ERR_INVALID_EXTEND_MODE

The value in `eci_extend_mode` field is not valid.

ECI_ERR_NO_CICS

Either the client or the server implementation is not available.

ECI_ERR_LUW_TOKEN

The value supplied in `eci_luw_token` is invalid.

ECI_ERR_THREAD_CREATE_ERROR

The server implementation or the client failed to create a thread to process the request.

ECI_ERR_ALREADY_ACTIVE

An attempt was made to continue an existing logical unit of work, but there was an outstanding asynchronous call for the same logical unit of work.

ECI_ERR_RESOURCE_SHORTAGE

The server implementation or the client did not have enough resources to complete the request.

ECI_ERR_NO_SESSIONS

A new logical unit of work was being created, but the application already has as many outstanding logical units of work as the configuration will support.

ECI_ERR_INVALID_DATA_AREA

Either the pointer to the ECI parameter block is invalid, or the pointer supplied in `eci_commarea` is invalid.

ECI_STATE_SYNC call type**Environment**

The `ECI_STATE_SYNC` call type is available in all environments.

Purpose

The ECI_STATE_SYNC call type provides a synchronous call that gives information about the status of the server.

ECI parameter block fields

The ECI parameter block should be set to nulls before setting the input parameter fields.

eci_call_type

Required input parameter.

Must be set to ECI_STATE_SYNC.

eci_commarea

Input parameter, required except when **eci_extend_mode** has the value ECI_STATE_CANCEL.

A pointer to the area of storage where the application receives the returned COMMAREA containing status information. See *Status information calls*, in the *External call interface* chapter, in *CICS Transaction Gateway: Programming Guide*, and “ECI status block” on page 131, for more details.

If **eci_extend_mode** has the value ECI_STATE_CANCEL, supply a null pointer and set the length (specified in **eci_commarea_length**) to zero.

eci_commarea_length

Required input and output parameter, except when **eci_extend_mode** has the value ECI_STATE_CANCEL.

The length of the COMMAREA in bytes, which must be the length of the ECI_STATUS structure that gives the layout of the status information COMMAREA. See *Status information calls*, in the *External call interface* chapter, in *CICS Transaction Gateway: Programming Guide*, and “ECI status block” on page 131, for more details. Area size must not exceed 32 500 bytes

If no COMMAREA is required, set this field to zero and supply a null pointer in **eci_commarea**.

reserved1

Output parameter.

This field was previously **eci_system_return_code**. In the CICS Transaction Gateway Version 3.1, and higher, this field is reserved for compatibility with earlier versions. No information is returned in this field; all system errors are written to the error log.

eci_extend_mode

Required input parameter.

An integer field further qualifying the call type. The values for this field (shown by their symbolic names) are as follows:

ECI_STATE_IMMEDIATE

Force a status reply to be sent as soon as it is available. The layout of the returned COMMAREA is defined in the ECI_STATUS structure. See *Status information calls*, in the *External call interface* chapter, in *CICS Transaction Gateway: Programming Guide*, and “ECI status block” on page 131, for more details.

ECI_STATE_CHANGED

Force a status reply to be sent only when the status changes. The supplied COMMAREA must contain the status as perceived by the application. A reply is sent only when there is a change from the status that the application supplied. The layout of the COMMAREA is defined in the ECI_STATUS structure. See *Status information calls*, in the *External call interface* chapter, in *CICS Transaction Gateway: Programming Guide*, and “ECI status block” on page 131, for more details. The **eci_luw_token** field that is returned on the immediate response provides a token to identify the request.

ECI_STATE_CANCEL

Cancel an ECI_STATE_CHANGED type of operation. No COMMAREA is required for this request. The **eci_luw_token** field must contain the token that was received during the ECI_STATE_CHANGED call.

eci_luw_token

Optional input and output parameter.

When a deferred status request is being set up (**eci_extend_mode** set to ECI_STATE_CHANGED), the token identifying the request is returned in the **eci_luw_token** field.

When a deferred status request is being cancelled (**eci_extend_mode** set to ECI_STATE_CANCEL), the **eci_luw_token** field must contain the token that was received during the ECI_STATE_CHANGED call.

This field is not used when other values of **eci_extend_mode** are specified.

eci_sysid

Required input parameter.

Reserved for future use, but this field should be initialized with nulls before the start of each logical unit of work.

eci_version

Required input parameter.

The version of the ECI for which the application is coded. Use the value ECI_VERSION_1A.

eci_system_name

Optional input parameter.

An 8-character field that specifies the name of the server for which status information is required. Pad unused characters with spaces. If supplied, it should be one of the server names returned by **CICS_EciListSystems**. The value may be supplied whenever **eci_luw_token** is set to zero.

If the field is set to nulls, the default server is selected; the name of the chosen server is returned in this field.

Return codes

See also the general list of return codes for **CICS_ExternalCall** in “CICS_ExternalCall ECI_Parms” on page 99.

ECI_NO_ERROR

The call completed successfully.

ECI_ERR_INVALID_DATA_LENGTH

The value in **eci_commarea_length** field is outside the valid range, or is inconsistent with the value in **eci_commarea**, being zero for a non-null **eci_commarea** pointer, or non-zero for a null **eci_commarea** pointer.

ECI_ERR_INVALID_EXTEND_MODE

The value in **eci_extend_mode** field is not valid.

ECI_ERR_LUW_TOKEN

The value supplied in **eci_luw_token** is invalid.

ECI_ERR_INVALID_DATA_AREA

Either the pointer to the ECI parameter block is invalid, or the pointer supplied in **eci_commarea** is invalid.

ECI_ERR_UNKNOWN_SERVER

The requested server could not be located. Only servers returned by **CICS_EciListSystems** are acceptable.

ECI_STATE_ASYNC call type**Environment**

Purpose

The `ECL_STATE_ASYNC` call type provides an asynchronous status information call. The calling application gets control back when the ECI accepts the request. At this point the parameters have been validated; however, the request might still be queued for later processing.

If no callback routine is provided, the application must use a reply solicitation call to determine that the request has ended and what the outcome was.

If a callback routine is provided, the callback routine `eci_callback` is invoked when a response is available.

Note: Some compilers do not support the use of callback routines. Consult your compiler documentation for more information.

Note: It is important that the `Eci` parameter blocks of outstanding `ECL_STATE_ASYNC` calls are not modified before the results of the call are received. Results will be incorrect if these blocks are modified before this stage.

When the callback routine is called, it is passed a single parameter—the value specified in `eci_message_qualifier`. This enables the callback routine to identify the asynchronous call that is completing. Note the following guidelines on the use of the callback routine:

1. The minimum possible processing should be performed within the callback routine.
2. ECI functions cannot be invoked from within the callback routine.
3. The callback routine should indicate to the main body of the application that the reply is available using an appropriate technique for the operating system upon which the ECI application is executing. For example, in a multithreaded environment, the callback routine might post a semaphore to signal another thread that an event has occurred.
4. The application, not the callback routine, must use a reply solicitation call to receive the actual response.

ECL parameter block fields

The ECI parameter block should be set to nulls before setting the input parameter fields.

`eci_call_type`

Required input parameter.

Must be set to `ECL_STATE_ASYNC`.

eci_commarea

Input parameter, required except when **eci_extend_mode** has the value `ECI_STATE_CANCEL`.

A pointer to the area of storage where the application receives the returned COMMAREA containing status information. See *Status information calls*, in the *External call interface* chapter, in *CICS Transaction Gateway: Programming Guide*, and “ECI status block” on page 131, for more details.

If **eci_extend_mode** has the value `ECI_STATE_CANCEL`, supply a null pointer and set the length (specified in **eci_commarea_length**) to zero.

eci_commarea_length

Required input parameter, except when **eci_extend_mode** has the value `ECI_STATE_CANCEL`.

The length of the COMMAREA in bytes, which must be the length of the `ECI_STATUS` structure that gives the layout of the status information COMMAREA. See *Status information calls*, in the *External call interface* chapter, in *CICS Transaction Gateway: Programming Guide*, and “ECI status block” on page 131, for more details. Area size must not exceed 32 500 bytes

If no COMMAREA is required, set this field to zero and supply a null pointer in **eci_commarea**.

reserved1

Output parameter.

This field was previously **eci_system_return_code**. In the CICS Transaction Gateway Version 3.1, and higher, this field is reserved for compatibility with earlier versions. No information is returned in this field; all system errors are written to the error log.

eci_extend_mode

Required input parameter.

An integer field further qualifying the call type. The values for this field (shown by their symbolic names) are as follows:

ECI_STATE_IMMEDIATE

Force a status reply to be sent immediately it is available. The layout of the returned COMMAREA is defined in the `ECI_STATUS` structure. See *Status information calls*, in the *External call interface* chapter, in *CICS Transaction Gateway: Programming Guide*, and “ECI status block” on page 131, for more details.

ECI_STATE_CHANGED

Force a status reply to be sent only when the status changes. The supplied COMMAREA must contain the status as perceived by the application. A reply is sent only when there is a change from the status that the application supplied. The layout of the COMMAREA is defined in the ECI_STATUS structure. See *Status information calls*, in the *External call interface* chapter, in *CICS Transaction Gateway: Programming Guide*, and “ECI status block” on page 131, for more details. The **eci_luw_token** field that is returned on the immediate response identifies the logical unit of work to which this call belongs.

ECI_STATE_CANCEL

Cancel an ECI_STATE_CHANGED type of operation. No COMMAREA is required for this request. The **eci_luw_token** field must contain the token that was received during the ECI_STATE_CHANGED call.

eci_message_qualifier

Optional input parameter.

An integer field allowing you to identify each asynchronous call if you are making more than one. If a callback routine is specified, the value in this field is returned to the callback routine during the notification process.

eci_luw_token

Optional input and output parameter.

When a deferred status request is being set up (**eci_extend_mode** set to ECI_STATE_CHANGED), the token identifying the request is returned in the **eci_luw_token** field.

When a deferred status request is being cancelled (**eci_extend_mode** set to ECI_STATE_CANCEL), the **eci_luw_token** field must contain the token that was received during the ECI_STATE_CHANGED call.

This field is not used when other values of **eci_extend_mode** are specified.

eci_sysid

Required input parameter.

Reserved for future use, but this field should be initialized with nulls before the start of each logical unit of work.

eci_version

Required input parameter.

The version of the ECI for which the application is coded. Use the value `ECI_VERSION_1A`.

eci_system_name

Optional input parameter.

An 8-character field that specifies the name of the server for which status information is requested. Pad unused characters with spaces. If supplied, it should be one of the server names returned by **CICS_EciListSystems**. The value may be supplied whenever **eci_luw_token** is set to zero.

If the field is set to nulls, the default server is selected. You can find out the name of the server from the **eci_system_name** field of the reply solicitation call you use to get the result of this asynchronous request. field.

eci_callback

Optional input parameter.

A pointer to the routine to be called when the asynchronous request completes. (The callback routine will be called only if the return code is `ECI_NO_ERROR`, and the pointer is not null.)

Return codes

See also the general list of return codes for **CICS_ExternalCall** in “CICS_ExternalCall ECI_Parms” on page 99.

If the return code is not `ECI_NO_ERROR`, the callback routine will not be called, and there will be no asynchronous reply for this request.

ECI_NO_ERROR

The call completed successfully.

ECI_ERR_INVALID_DATA_LENGTH

The value in **eci_commarea_length** field is outside the valid range, or is inconsistent with the value in **eci_commarea**, being zero for a non-null **eci_commarea** pointer, or non-zero for a null **eci_commarea** pointer.

ECI_ERR_INVALID_EXTEND_MODE

The value in **eci_extend_mode** field is not valid.

ECI_ERR_LUW_TOKEN

The value supplied in **eci_luw_token** is invalid.

ECI_ERR_INVALID_DATA_AREA

Either the pointer to the ECI parameter block is invalid, or the pointer supplied in **eci_commarea** is invalid.

ECI_GET_REPLY call type

Purpose

The ECI_GET_REPLY call type provides a reply solicitation call to return information appropriate to any outstanding reply for any asynchronous request. If there is no such reply, ECI_ERR_NO_REPLY is returned. (To cause the application to wait until a reply is available, use call type ECI_GET_REPLY_WAIT instead.)

Note: It is important that the Eci parameter blocks of outstanding ECI_ASYNC calls are not modified before the results of the call are received (for example using this get reply call). Results will be incorrect if these blocks are modified before this stage.

ECI parameter block fields

The ECI parameter block should be set to nulls before setting the input parameter fields.

The following fields are the fields of the ECI parameter block that might be supplied as input.

In the course of an ECI_GET_REPLY call, the ECI parameter block is updated as follows:

1. All the outputs from the reply, some of which overwrite input fields, are added. These fields are those that are output from the corresponding synchronous version of the asynchronous request.
2. The **eci_message_qualifier** value supplied as input to the asynchronous request to which this reply relates is restored.
3. Any inputs that are not updated become undefined, except the pointer to the COMMAREA. Do not use the contents of these fields again.

eci_call_type

Required input parameter.

Must be set to ECI_GET_REPLY.

eci_commarea

Optional input parameter.

A pointer to the area of storage where the application receives the returned COMMAREA. The contents of the returned commarea depend on the type of asynchronous call to which a reply is being sought. For a program link call, it is the COMMAREA expected to be returned from the called program, if any. For a status information call, except when **eci_extend_mode** has the value ECI_STATE_CANCEL, it

is a COMMAREA containing status information. See *Status information calls*, in the *External call interface* chapter, in *CICS Transaction Gateway: Programming Guide*, and “ECI status block” on page 131, for more details.

If no COMMAREA is required, supply a null pointer and set the length (specified in **eci_commarea_length**) to zero.

If the code page of the application is different from the code page of the server, data conversion must be performed at the server. To do this, use CICS-supplied resource conversion capabilities, such as the DFHCNV macro definitions.

eci_commarea_length

Required input parameter.

The length of the COMMAREA in bytes. This value may not exceed 32 500. (Some client/server combinations may allow larger COMMAREAs, but this is not guaranteed to work.)

If no COMMAREA is required, set this field to zero and supply a null pointer in **eci_commarea**.

eci_sysid

Required input parameter.

Reserved for future use, but this field should be initialized with nulls before the start of each logical unit of work.

eci_version

Required input parameter.

The version of the ECI for which the application is coded. Use the value ECI_VERSION_1A.

Return codes

See also the general list of return codes for **CICS_ExternalCall** in “CICS_ExternalCall ECI_Parms” on page 99.

ECI_NO_ERROR

The asynchronous request to which this reply relates completed successfully.

ECI_ERR_INVALID_DATA_LENGTH

The value in **eci_commarea_length** field is unacceptable for one of the following reasons:

- It is outside the valid range.
- It is inconsistent with the value in **eci_commarea**, being zero for a non-null **eci_commarea** pointer, or non-zero for a null **eci_commarea** pointer.

- It is not large enough for the output COMMAREA from the asynchronous request to which this reply relates.

In the last case, you can use the output **eci_commarea_length** to allocate more storage for the COMMAREA, and then use the output **eci_message_qualifier** (if it identifies the asynchronous request uniquely) with an ECI_GET_SPECIFIC_REPLY call type to retrieve the reply.

ECI_ERR_NO_CICS

The CICS server specified in **eci_system_name** in the asynchronous request to which this reply relates is not available. No resources have been updated.

ECI_ERR_CICS_DIED

A logical unit of work was to be begun or continued by the asynchronous request to which this reply relates, but the CICS server was no longer available. If **eci_extend_mode** was ECI_EXTENDED, the changes are backed out, and the logical unit of work ends. If **eci_extend_mode** was ECI_NO_EXTEND, ECI_COMMIT, or ECI_BACKOUT, the application cannot determine whether the changes have been committed or backed out, and must log this condition to aid future manual recovery.

ECI_ERR_NO_REPLY

There was no outstanding reply.

ECI_ERR_TRANSACTION_ABEND

The asynchronous request to which this reply relates caused a program to be executed in the server, but the CICS transaction that executed the requested program abended. The abend code will be found in **eci_abend_code**. For information about abend codes and their meaning, consult the documentation for the server system to which the request was directed.

ECI_ERR_THREAD_CREATE_ERROR

The CICS server or CICS Transaction Gateway failed to create the thread to process the asynchronous call to which this reply relates.

ECI_ERR_RESOURCE_SHORTAGE

The server implementation or CICS Transaction Gateway did not have enough resources to complete the asynchronous request to which this reply relates.

ECI_ERR_INVALID_DATA_AREA

Either the pointer to the ECI parameter block is invalid, or the pointer supplied in **eci_commarea** is invalid.

ECI_ERR_ROLLEDBACK

The asynchronous request to which this reply relates attempted to

commit a logical unit of work, but the server was unable to commit the changes, and backed them out instead.

ECI_ERR_UNKNOWN_SERVER

The asynchronous request to which this reply relates specified a server that could not be located. Only servers returned by **CICS_EciListSystems** are acceptable.

ECI_ERR_MAX_SESSIONS

There were not enough communication resources to satisfy the asynchronous request to which this reply relates. Consult the documentation for your CICS Transaction Gateway or server to see how to control communication resources.

ECI_ERR_MAX_SYSTEMS

The asynchronous request to which this reply relates attempted to start requests to more servers than your configuration allows. Consult the documentation for your CICS Transaction Gateway or server to see how to control the number of servers you can use.

ECI_ERR_SECURITY_ERROR

You did not supply a valid combination of userid and password on the asynchronous request to which this reply relates.

ECI_GET_REPLY_WAIT call type

Purpose

The **ECI_GET_REPLY_WAIT** call type provides a reply solicitation call to return information appropriate to any outstanding reply for any asynchronous request. If there is no such reply, the application waits until there is. (You can get an indication that no reply is available by using call type **ECI_GET_REPLY** instead.)

Note: It is important that the **Eci** parameter blocks of outstanding **ECI_STATE_ASYNC** calls are not modified before the results of the call are received. Results will be incorrect if these blocks are modified before this stage.

ECI parameter block fields

Same as for **ECI_GET_REPLY**, but **eci_call_type** must be set to **ECI_GET_REPLY_WAIT**.

Return codes

Same as for **ECI_GET_REPLY**, except that **ECI_ERR_NO_REPLY** cannot be returned.

ECI_GET_SPECIFIC_REPLY call type

Purpose

The ECI_GET_SPECIFIC_REPLY call type provides a reply solicitation call to return information appropriate to any outstanding reply that matches the **eci_message_qualifier** input. If there is no such reply, ECI_ERR_NO_REPLY is returned. (To cause the application to wait until a reply is available, use call type ECI_GET_REPLY_WAIT instead.)

Note: It is important that the Eci parameter blocks of outstanding ECI_STATE_ASYNC calls are not modified before the results of the call are received. Results will be incorrect if these blocks are modified before this stage.

ECI parameter block fields

Set the ECI parameter block to nulls before setting the input parameter fields.

The following fields are the fields of the ECI parameter block that might be supplied as input.

In the course of an ECI_GET_REPLY call, the ECI parameter block is updated as follows:

1. All the outputs from the reply, some of which overwrite input fields, are added. These fields are those that are output from the corresponding synchronous version of the asynchronous request.
2. Any inputs that are not updated become undefined, except the pointer to the COMMAREA and the input **eci_message_qualifier**. Do not use the contents of these fields again.

eci_call_type

Required input parameter.

Must be set to ECI_GET_SPECIFIC_REPLY.

eci_commarea

Optional input parameter.

A pointer to the area of storage where the application receives the returned COMMAREA. The contents of the returned commarea depend on the type of asynchronous call to which a reply is being sought. For a program link call, it is the COMMAREA expected to be returned from the called program, if any. For a status information call, except one in which **eci_extend_mode** had the value ECI_STATE_CANCEL, it is a COMMAREA containing status information. See *Status information calls*, in the *External call interface*

chapter, in *CICS Transaction Gateway: Programming Guide*, and “ECI status block” on page 131, for more details.

If the code page of the application is different from the code page of the server, data conversion must be performed at the server. To do this, use CICS-supplied resource conversion capabilities, such as the DFHCNV macro definitions.

eci_commarea_length

Required input parameter.

The length of the COMMAREA in bytes. This value may not exceed 32 500. (Some client/server combinations may allow larger COMMAREAs, but this is not guaranteed to work.)

eci_message_qualifier

Required input parameter.

An integer field that identifies the asynchronous call for which a reply is being solicited.

eci_sysid

Required input parameter.

Reserved for future use, but this field should be initialized with nulls before the start of each logical unit of work.

eci_version

Required input parameter.

The version of the ECI for which the application is coded. Use the value ECI_VERSION_1A.

Return codes

See also the general list of return codes for **CICS_ExternalCall** in “CICS_ExternalCall ECI_Parms” on page 99.

ECI_NO_ERROR

The call completed successfully.

ECI_ERR_INVALID_DATA_LENGTH

The value in **eci_commarea_length** field is unacceptable for one of the following reasons:

- It is outside the valid range.
- It is inconsistent with the value in **eci_commarea**, being zero for a non-null **eci_commarea** pointer, or non-zero for a null **eci_commarea** pointer.
- It is not large enough for the output COMMAREA from the asynchronous request to which this reply relates.

In the last case, you can use the output **eci_commarea_length** to allocate more storage for the COMMAREA, and then retry the ECI_GET_SPECIFIC_REPLY call.

ECI_ERR_NO_CICS

The CICS server specified in **eci_system_name** in the asynchronous request to which this reply relates is not available. No resources have been updated.

ECI_ERR_CICS_DIED

A logical unit of work was to be begun or continued by the asynchronous request to which this reply relates, but the CICS server was no longer available. If **eci_extend_mode** was ECI_EXTENDED, the changes are backed out, and the logical unit of work ends. If **eci_extend_mode** was ECI_NO_EXTEND, ECI_COMMIT, or ECI_BACKOUT, the application cannot determine whether the changes have been committed or backed out, and must log this condition to aid future manual recovery.

ECI_ERR_NO_REPLY

There was no outstanding reply that matched the input **eci_message_qualifier**.

ECI_ERR_TRANSACTION_ABEND

The asynchronous request to which this reply relates caused a program to be executed in the server, but the CICS transaction that executed the requested program abended. The abend code will be found in **eci_abend_code**. For information about abend codes and their meaning, consult the documentation for the server system to which the request was directed.

ECI_ERR_THREAD_CREATE_ERROR

The CICS server or CICS Transaction Gateway failed to create the thread to process the asynchronous request to which this reply relates.

ECI_ERR_RESOURCE_SHORTAGE

The CICS server or CICS Transaction Gateway did not have enough resources to complete the asynchronous request to which this reply relates.

ECI_ERR_INVALID_DATA_AREA

Either the pointer to the ECI parameter block is invalid, or the pointer supplied in **eci_commarea** is invalid.

ECI_ERR_ROLLEDBACK

The asynchronous request to which this reply relates attempted to commit a logical unit of work, but the server was unable to commit the changes, and backed them out instead.

ECI_ERR_UNKNOWN_SERVER

The asynchronous request to which this reply relates specified a server that could not be located. Only servers returned by **CICS_EciListSystems** are acceptable.

ECI_ERR_MAX_SESSIONS

There were not enough communication resources to satisfy the asynchronous request to which this reply relates. Consult the documentation for your CICS Transaction Gateway or server to see how to control communication resources.

ECI_ERR_MAX_SYSTEMS

The asynchronous request to which this reply relates attempted to start requests to more servers than your configuration allows. Consult the documentation for your CICS Transaction Gateway or server to see how to control the number of servers you can use.

ECI_ERR_SECURITY_ERROR

You did not supply a valid combination of userid and password on the asynchronous request to which this reply relates.

ECI_GET_SPECIFIC_REPLY_WAIT call type

Environment

Purpose

The **ECI_GET_SPECIFIC_REPLY_WAIT** call type provides a reply solicitation call to return information appropriate to any outstanding reply that matches the input **eci_message_qualifier**. If there is no such reply, the application waits until there is. (You can get an indication that no reply is available by using call type **ECI_GET_SPECIFIC_REPLY** instead.)

Note: It is important that the Eci parameter blocks of outstanding **ECI_STATE_ASYNC** calls are not modified before the results of the call are received. Results will be incorrect if these blocks are modified before this stage.

ECI parameter block fields

Same as for **ECI_GET_SPECIFIC_REPLY**, but **eci_call_type** must be set to **ECI_GET_SPECIFIC_REPLY_WAIT**.

Return codes

Same as for **ECI_GET_SPECIFIC_REPLY**, except that **ECI_ERR_NO_REPLY** cannot be returned.

Note: If you issue an `ECI_GET_SPECIFIC_REPLY_WAIT` call against an outstanding `ECI_STATE_AYSNC` call with `eci_extend mode` set to `ECI_STATE_CHANGED`, no response will ever be received if an `ECI_STATE_ASYNC` call with `eci_extend mode` set to `ECI_STATE_CANCEL` is issued.

ECI status block

The ECI status block is used in status information calls to pass information to and from the ECI. It contains the following fields:

ConnectionType

An integer field specifying the type of system on which the application is running, with the following possible values:

ECI_CONNECTED_NOWHERE

Application is not connected to anything.

ECI_CONNECTED_TO_CLIENT

Application is running on a client system.

ECI_CONNECTED_TO_SERVER

Application is using a server implementation of the ECI.

CicsServerStatus

An integer field specifying the state of the CICS server, with the following possible values:

ECI_SERVERSTATE_UNKNOWN

The CICS server state could not be determined.

ECI_SERVERSTATE_UP

The CICS server is available to run programs.

ECI_SERVERSTATE_DOWN

The CICS server is not available to run programs.

CicsClientStatus

An integer field specifying the state of the Client daemon, with the following possible values:

ECI_CLIENTSTATE_UNKNOWN

The Client daemon state could not be determined.

ECI_CLIENTSTATE_UP

The Client daemon is available to receive ECI calls.

ECI_CLIENTSTATE_INAPPLICABLE

The application is using a server implementation of the ECI.

Purpose

The **CICS_EciListSystems** function provides a list of CICS servers to which **CICS_ExternalCall** requests may be directed. There is no guarantee that a communications link exists between the Client daemon and any server in the list, or that any of the servers is available to process requests.

The list of servers is returned as an array of system information structures, one element for each CICS server. The structure, called **CICS_EciSystem_t**, defines the following fields.

SystemName

A pointer to a null-terminated string specifying the name of a CICS server. If the name is shorter than **CICS_ECI_SYSTEM_MAX**, it is padded with nulls to a length of **CICS_ECI_SYSTEM_MAX + 1**.

Description

A pointer to a null-terminated string that provides a description of the system, if one is available. If the description is shorter than **CICS_ECI_DESCRIPTION_MAX** characters, it is padded with nulls to a length of **CICS_ECI_DESCRIPTION_MAX + 1**.

Parameters**NameSpace**

A pointer reserved for future use. Ensure that this is a null pointer.

Systems

On entry to the function, this parameter specifies the number of elements in the array provided in the **List** parameter. On return it contains the actual number of systems found.

List An array of **CICS_EciSystem_t** structures that are filled in and returned by the function. The application must provide storage for the array, and must set the **Systems** parameter to indicate the number of elements in the array. The first name in the list is the default server. However, the way in which the default is defined depends upon the operating system.

Return Codes**ECI_NO_ERROR**

The function completed successfully. The number of systems found is at least one, and does not exceed the value supplied as input in the **Systems** parameter.

ECI_ERR_MORE_SYSTEMS

There was not enough space in the **List** array to store the information. The supplied array has been filled, and the **Systems** parameter has

been updated to contain the total number of systems found, so that you can reallocate an array of suitable size and try the function again.

ECI_ERR_NO_SYSTEMS

No CICS servers can be located. In this case, the value returned in **Systems** is zero.

ECI_ERR_NO_CICS

The Client daemon is not active.

ECI_ERR_INVALID_DATA_LENGTH

The value specified in the **Systems** parameter is so large that the length of storage for the **List** parameter exceeds 32 767.

ECI_ERR_CALL_FROM_CALLBACK

The call was made from a callback routine.

ECI_ERR_SYSTEM_ERROR

An internal system error occurred.

External Presentation Interface

EPI constants and data structures

This section describes the constants and data structures that you will need to use the EPI. They are referred to in “EPI functions” on page 139.

EPI constants

The following constants are referred to symbolically in the descriptions of the EPI data structures, functions, and events in this Chapter. Their values are given here to help you understand the descriptions. However, your code should always use the symbolic names of EPI constants provided for the programming language you are using.

Lengths of fields

- CICS_EPI_SYSTEM_MAX (8)
- CICS_EPI_DESCRIPTION_MAX (60)
- CICS_EPI_NETNAME_MAX (8)
- CICS_EPI_TRANSID_MAX (4)
- CICS_EPI_ABEND_MAX (4)
- CICS_EPI_DEVTYPE_MAX (16)
- CICS_EPI_ERROR_MAX (60).
- CICS_EPI_PASSWORD_MAX (10)
- CICS_EPI_USERID_MAX (10)
- CICS_EPI_MAPNAME_MAX (7)
- CICS_EPI_MAPSETNAME_MAX (8)

- CICS_EPI_TERMID_MAX (4)

Relating to TermIndex

- CICS_EPI_TERM_INDEX_NONE 0xFFFF.

Version numbers (See *EPI versions*, in *CICS Transaction Gateway: Programming Guide*.)

- CICS_EPI_VERSION_200

EPI data structures

The following data structures are available for use with the EPI.

- CICS_EpiSystem_t
- CICS_EpiAttributes_t
- CICS_EpiDetails_t
- CICS_EpiEventData_t

In the descriptions of the fields in the data structures, fields described as strings are null-terminated strings.

CICS_EpiSystem_t:

Purpose

The **CICS_EpiSystem_t** structure contains the name and description of a CICS server. An array of these structures is returned from the **CICS_EpiListSystems** function.

Fields

SystemName

A string naming the CICS server. It can be passed as a parameter to the **CICS_EpiAddTerminal** and **CICS_EpiAddExTerminal** functions, to identify the CICS server in which the terminal resource should be installed. If the name is shorter than CICS_EPI_SYSTEM_MAX characters, it is padded with nulls to a length of CICS_EPI_SYSTEM_MAX + 1.

Description

A string giving a brief description of the server. If the description is shorter than CICS_EPI_DESCRIPTION_MAX, it is padded with nulls to a length of CICS_EPI_DESCRIPTION_MAX + 1.

CICS_EpiAttributes_t:

Purpose

The `CICS_EpiAttributes_t` structure holds information about the attributes to be associated with a terminal resource installed by the `CICS_EpiAddExTerminal` function.

Fields

EpiAddType

Indicates whether the application is prepared to wait until the request to install the terminal is complete. Use one of the following values:

`CICS_EPI_ADD_ASYNC`

The calling application gets control back when the request to install the terminal resource has been accepted; at this point the parameters have been validated.

Assuming valid parameters, the `CICS_EPI_EVENT_ADD_TERM` event is generated when the request to install the terminal has completed.

The `TermIndex` is returned for use with the `CICS_EpiGetEvent` function.

`CICS_EPI_ADD_SYNC`

The calling application gets control back when the request to install the terminal resource has completed. Returned information is immediately available.

InstallTimeOut

A value in the range 0 through 3600, specifying the maximum time in seconds that installation of the terminal resource is allowed to take; a value of 0 means that no limit is set.

A value of 3600 is assumed if a larger value is specified.

ReadTimeOut

A value in the range 0 through 3600, specifying the maximum time in seconds that is allowed between notification of a `CICS_EPI_EVENT_CONVERSE` event for the terminal resource and the following invocation of the `CICS_EpiReply`; a value of 0 means that no limit is set.

A value of 3600 is assumed if a larger value is specified.

If time-out occurs, the conversation is abended. This results in a `CICS_EPI_EVENT_END_TRAN` event being generated; the `EndReason` field is set to `CICS_EPI_READTIMEOUT_EXPIRED`; the `AbendCode` field is not set.

SignonCapability

Indicates whether the application may start server-provided sign-on and signoff transactions from the terminal resource. Use one of the following values:

CICS_EPI_SIGNON_CAPABLE

The terminal resource is to be installed as sign-on capable.

CICS_EPI_SIGNON_INCAPABLE

The resource is to be installed as sign-on incapable.

CCSID

A value in the range 1 through 65536 specifying the coded character set identifier (CCSID) that identifies the coded graphic character set used by the client application for data passed between the terminal resource and CICS transactions.

A value of 0 means that a default CCSID is used.

For details on the CCSID values for various character sets see *Data conversion when using the Client daemon*, in the *CICS Transaction Gateway: Administration* book for your operating system.

UserId

A string specifying the userid to be associated with the terminal resource. If the userid is shorter than CICS_EPI_USERID_MAX, it must be padded with nulls to a length of CICS_EPI_USERID_MAX+1.

Password

A string specifying the password to be associated with the terminal resource. If the password is shorter than CICS_EPI_PASSWORD_MAX characters, it must be padded with nulls to a length of CICS_EPI_PASSWORD_MAX+1.

CICS_EpiDetails_t:

Purpose

The **CICS_EpiDetails_t** structure holds information about a terminal resource installed by the **CICS_EpiAddTerminal** or the **CICS_EpiAddExTerminal** function.

Fields

NetName

A string specifying the VTAM[®]-style netname of the terminal resource. If the name is shorter than CICS_EPI_NETNAME_MAX characters, it is padded with nulls to a length of CICS_EPI_NETNAME_MAX + 1.

NumLines

The number of rows supported by the terminal resource.

NumColumns

The number of columns supported by the terminal resource.

MaxData

The maximum size of data that can be sent to this terminal resource from a CICS transaction, and the maximum size of data that can be sent from this terminal resource to a CICS transaction by a **CICS_EpiStartTran** call or **CICS_EpiReply** call.

The maximum size may be defined in the model terminal definition specified by the **DevType** parameter on the **CICS_EpiAddTerminal** call that installed the terminal resource in the server. If the value either is not specified in the model terminal definition, a default value of 12000 is assumed.

ErrLastLine

1 if the terminal resource should display error messages on its last row, 0 otherwise.

ErrIntensify

1 if the terminal resource should display error messages intensified, 0 otherwise.

ErrColor

The 3270 attribute defining the color to be used to display error messages.

ErrHighlight

The 3270 attribute defining the highlight value to be used to display error messages.

Hilight

1 if the terminal resource is defined to support extended highlighting, 0 otherwise.

Color 1 if the terminal resource is defined to support color, 0 otherwise.

System

A string specifying the name of the server in which the terminal resource has been installed. If the name is shorter than **CICS_EPI_SYSTEM_MAX** characters, it is padded with nulls to a length of **CICS_EPI_SYSTEM_MAX + 1**.

TermId

A string specifying the name of the terminal resource. If the name is shorter than **CICS_EPI_TERMID_MAX** characters, it is padded with nulls to a length of **CICS_EPI_TERMID_MAX + 1**.

SignonCapability

The sign-on capability assigned by the server to the terminal resource:

CICS_EPI_SIGNON_CAPABLE

if the application may start server-provided sign-on and signoff transactions at the terminal resource.

CICS_EPI_SIGNON_INCAPABLE

if the application may not start server-provided sign-on and signoff transactions at the terminal resource.

CICS_EPI_SIGNON_UNKNOWN

if the **CICS_EpiAddTerminal** function was used to add the terminal resource. (This value is also returned if the **CICS_EpiAddExTerminal** function was used to add the terminal resource and prerequisite changes have not been applied to the server.)

CICS_EpiEventData_t:

Purpose

The **CICS_EpiEventData_t** structure holds details of a terminal-related event. Not all fields are valid for all events, and fields that are not valid are set to nulls. This structure is an output from **CICS_EpiGetEvent**.

Fields

TermIndex

The terminal index for the terminal resource against which this event occurred.

Event The event indicator; that is, one of the event codes listed in “EPI events” on page 166.

EndReason

The reason for termination, if the event is a **CICS_EPI_EVENT_END_TERM** or **CICS_EPI_EVENT_END_TRAN** event.

TransId

A string specifying a transaction name. If the name is shorter than **CICS_EPI_TRANSID_MAX** characters, it is padded with spaces to this length, followed by a single null character.

Reserved1

A reserved field.

Prior to CICS Transaction Gateway Version 3.1, this field was called **AbendCode**.

Data A pointer to a buffer that is updated with any terminal data stream associated with the event.

On input the **Data** parameter should be set to point to a **CICS_EpiDetails_t** structure on the first invocation of **CICS_EpiGetEvent** for a terminal being added asynchronously. The details structure is updated on return from **CICS_EpiGetEvent**.

Size The maximum size of the buffer addressed by **Data**. On return from the **CICS_EpiGetEvent** call, this contains the actual length of data returned.

EndReturnCode

A string containing the CICS_EPI_returncode.

MapName

A string specifying the name of the map that was most recently referenced in the MAP option of a SEND MAP command processed for the terminal resource, if the event is a CICS_EPI_EVENT_SEND or a CICS_EPI_EVENT_CONVERSE event. If the terminal resource is not supported by BMS, or the server has no record of any map being sent, the value returned is spaces. If the name is shorter than CICS_EPI_MAPNAME_MAX characters, it is padded with spaces to this length, followed by a single null character.

MapSetName

A string specifying the name of the mapset that was most recently referenced in the MAPSET option of a SEND MAP command processed for the terminal resource, if the event is a CICS_EPI_EVENT_SEND or a CICS_EPI_EVENT_CONVERSE event. If the MAPSET option was not specified on the most recent request, BMS used the map name as the mapset name. In both cases, the mapset name used may have been suffixed by a terminal suffix. If the terminal resource is not supported by BMS, or the server has no record of any mapset being sent, the value returned is spaces. If the name is shorter than CICS_EPI_MAPSETNAME_MAX characters, it is padded with spaces to this length, followed by a single null character.

Note: The **Data** and **Size** fields should be set before the call to **CICS_EpiGetEvent** is made.

EPI functions

This section describes the functions provided by the EPI that can be called from an application program:

- **CICS_EpiInitialize**
- **CICS_EpiTerminate**
- **CICS_EpiListSystems**
- **CICS_EpiAddTerminal**

- CICS_EpiAddExTerminal
- CICS_EpiInquireSystem
- CICS_EpiDelTerminal
- CICS_EpiPurgeTerminal
- CICS_EpiSetSecurity
- CICS_EpiStartTran
- CICS_EpiReply
- CICS_EpiATISate
- CICS_EpiGetEvent

Table 1 summarizes the functions of the interface, the parameters passed to each function, and the possible return codes from each function.

The mapping of actual return code values to the symbolic names is contained in the following file for the Windows operating systems:

```
C      \include\cics_eci.h
Cobol \copybook\cicsepi.cbl
```

and in the following files for the UNIX and Linux operating systems:

```
C      /include/cics_eci.h
```

Table 1. Summary of EPI functions

Function name	Parameters	Return codes: CICS_EPI_
CICS_EpiInitialize	Version	ERR_FAILED ERR_IS_INIT ERR_VERSION NORMAL
CICS_EpiTerminate	none	ERR_FAILED ERR_NOT_INIT ERR_IN_CALLBACK NORMAL
CICS_EpiListSystems	NameSpace Systems List	ERR_FAILED ERR_MORE_SYSTEMS ERR_NO_SYSTEMS ERR_NOT_INIT ERR_NULL_PARM ERR_IN_CALLBACK NORMAL

Table 1. Summary of EPI functions (continued)

Function name	Parameters	Return codes: CICS_EPI_
CICS_EpiAddTerminal	NameSpace System Netname DevType NotifyFn Details TermIndex	ERR_ALREADY_INSTALLED ERR_FAILED ERR_IN_CALLBACK ERR_MAX_SESSIONS ERR_MAX_SYSTEMS ERR_MODELID_INVALID ERR_NOT_3270_DEVICE ERR_NOT_INIT ERR_NULL_PARM ERR_RESOURCE_SHORTAGE ERR_SECURITY ERR_SERVER_BUSY ERR_SERVER_DOWN ERR_SYSTEM ERR_TERMID_INVALID NORMAL
CICS_EpiAddExTerminal	System Netname DevType NotifyFn Details TermIndex Attributes	ERR_FAILED ERR_NOT_INIT ERR_SYSTEM ERR_SECURITY ERR_NULL_PARM ERR_VERSION ERR_IN_CALLBACK ERR_SERVER_DOWN ERR_RESPONSE_TIMEOUT ERR_SIGNON_NOT_POSS ERR_PASSWORD_INVALID ERR_ADDTYPE_INVALID ERR_SIGNONCAP_INVALID ERR_USERID_INVALID ERR_TERMID_INVALID ERR_MODELID_INVALID ERR_NOT_3270_DEVICE ERR_ALREADY_INSTALLED ERR_CCSID_INVALID ERR_SERVER_BUSY ERR_RESOURCE_SHORTAGE ERR_MAX_SESSIONS ERR_MAX_SYSTEMS NORMAL

Table 1. Summary of EPI functions (continued)

Function name	Parameters	Return codes: CICS_EPI_
CICS_EpiInquireSystem	TermIndex System	ERR_BAD_INDEX ERR_FAILED ERR_NOT_INIT ERR_NULL_PARM ERR_IN_CALLBACK NORMAL
CICS_EpiDelTerminal	TermIndex	ERR_BAD_INDEX ERR_FAILED ERR_NOT_INIT ERR_TRAN_ACTIVE ERR_IN_CALLBACK NORMAL
CICS_EpiPurgeTerminal	TermIndex	ERR_BAD_INDEX ERR_FAILED ERR_NOT_INIT ERR_IN_CALLBACK ERR_VERSION NORMAL
CICS_EpiSetSecurity	TermIndex UserId Password	ERR_NOT_INIT ERR_BAD_INDEX ERR_IN_CALLBACK ERR_SYSTEM_ERROR ERR_VERSION ERR_PASSWORD_INVALID ERR_USERID_INVALID ERR_NULL_PASSWORD ERR_NULL_USERID NORMAL
CICS_EpiStartTran	TermIndex TransId Data Size	ERR_ATL_ACTIVE ERR_BAD_INDEX ERR_FAILED ERR_NO_DATA ERR_NOT_INIT ERR_TTI_ACTIVE ERR_IN_CALLBACK ERR_SERVER_DOWN ERR_RESOURCE_SHORTAGE ERR_MAX_SESSIONS NORMAL

Table 1. Summary of EPI functions (continued)

Function name	Parameters	Return codes: CICS_EPI_
CICS_EpiReply	TermIndex Data Size	ERR_BAD_INDEX ERR_FAILED ERR_NO_CONVERSE ERR_NO_DATA ERR_NOT_INIT ERR_IN_CALLBACK ERR_ABENDED ERR_SERVER_DOWN NORMAL
CICS_EpiATISate	TermIndex ATISate	ERR_ATI_STATE ERR_BAD_INDEX ERR_FAILED ERR_NOT_INIT ERR_IN_CALLBACK ERR_NULL_PARAM NORMAL
CICS_EpiGetEvent	TermIndex Wait	ERR_BAD_INDEX ERR_FAILED ERR_MORE_DATA ERR_MORE_EVENTS ERR_NO_EVENT ERR_NOT_INIT ERR_WAIT ERR_NULL_PARAM ERR_IN_CALLBACK NORMAL
CICS_GetSysError	TermIndex SysErr	ERR_NOT_INIT ERR_BAD_INDEX ERR_FAILED ERR_NULL_PARAM ERR_VERSION NORMAL

Refer to the definitions of the functions to discover the types and usage of the parameters, the data structures used by the functions, and the meanings of the return codes.

CICS_EpiInitialize

CICS_EpiInitialize	Version
--------------------	---------

Purpose

The **CICS_EpiInitialize** function initializes the EPI. All other EPI calls from this application are invalid before this call is made.

Parameters

Version

The version of the EPI for which this application is coded. This makes it possible for old applications to remain compatible with future versions of the EPI. The version described here is **CICS_EPI_VERSION_200**. See *EPI versions*, in *CICS Transaction Gateway: Programming Guide*, for more information.

The EPI uses this parameter only for input.

Return codes

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_IS_INIT

The EPI is already initialized.

CICS_EPI_ERR_VERSION

The EPI cannot support the version requested.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiTerminate

CICS_EpiTerminate

Purpose

The **CICS_EpiTerminate** function ends the application's use of the EPI, typically just before the application terminates. All other EPI calls (except for **CICS_EpiInitialize**) are invalid when this call has completed.

The application should issue **CICS_EpiDelTerminal** calls before terminating, to delete any terminal resources.

Parameters

None.

Return codes

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_TTI_ACTIVE

A transaction started from the EPI is still active or a CICS_EpiGetEvent call is still outstanding.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiListSystems

CICS_EpiListSystems	Namespace Systems List
---------------------	------------------------------

Purpose

The **CICS_EpiListSystems** function returns a list of CICS servers that are candidates to act as servers for EPI requests. There is no guarantee that a communications link exists between the CICS Transaction Gateway and any server in the list, or that any of the servers is available to process requests.

The list is returned as an array of system information structures, one element for each CICS server. See “CICS_EpiSystem_t” on page 134 for the contents of the structure.

EPI applications should call this function immediately after each **CICS_EpiInitialize** call made to determine which CICS servers are available.

Parameters

Namespace

A pointer reserved for future use. Ensure that this is a null pointer.

Systems

A pointer to a number. On entry to the function, this number specifies the number of elements in the array specified in the **List** parameter.

This value should accurately reflect the amount of storage that is available to the EPI to store the result. On return, it contains the actual number of servers found.

The EPI uses this parameter for both input and output.

List An array of **CICS_EpiSystem_t** structures that are filled in and returned by the function. The application should provide the storage for the array and must set the **Systems** parameter to indicate the number of elements in the array. The first name in the list is the default server. However, the way in which the default is defined is operating system dependent.

The EPI uses this parameter only for output.

Return codes

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_MORE_SYSTEMS

There was not enough space in the **List** array to store the details of all the CICS servers found. The supplied array has been filled, and the **Systems** parameter has been updated to contain the total number of servers found, thus allowing you to reallocate an array of suitable size and try the function again.

CICS_EPI_ERR_NO_SYSTEMS

No CICS servers can be located. In this case, the value returned in **Systems** is zero.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_NULL_PARM

Systems is a null pointer.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NORMAL

The function completed successfully. The number of systems found is at least one, and does not exceed the value supplied as input in the **Systems** parameter.

CICS_EpiAddTerminal

The **CICS_EpiAddTerminal** function installs a new terminal resource, or reserves an existing terminal resource, for the application's use.

CICS_EpiAddTerminal

Namespace
System
NetName
DevType
NotifyFn
Details
TermIndex

Purpose

It provides a terminal index, which should be used to identify the terminal resource on all further EPI calls. It also provides the information defined in the **CICS_EpiDetails_t** data structure.

There is a limit on the number of terminals you can add with this operation: The maximum varies according to the resources available on the client system.

Note: The **CICS_EpiAddTerminal** function adds terminal resources whose sign-on capability is dependant upon the server in which the terminal resource is installed, for example, they would be sign-on incapable on CICS Transaction Server for z/OS[®] servers.

Parameters

Namespace

A pointer reserved for future use. Ensure that this is a null pointer.

System

A pointer to a null-terminated string that specifies the name of the server in which the terminal resource is to be installed or reserved. If the name is shorter than **CICS_EPI_SYSTEM_MAX** characters, it must be padded with nulls to a length of **CICS_EPI_SYSTEM_MAX + 1**.

If the string is all nulls, the default server is selected by the EPI. To determine the name of the server chosen, use **CICS_EpiInquireSystem**.

The EPI uses this parameter only for input.

NetName

A pointer to a null-terminated string that specifies the name of the terminal resource to be installed or reserved, or null. The interpretation of this name is server-dependent.

If a string is supplied that is shorter than **CICS_EPI_NETNAME_MAX**, it must be padded with nulls to a length of **CICS_EPI_NETNAME_MAX + 1**.

The string is transmitted to the server without conversion to uppercase.

The characters used are translated from the client's code page to an EBCDIC code page before transmission. If the server uses an ASCII code page, they will be retranslated. The only characters guaranteed to be invariant under these translations are the uppercase characters A to Z, and the numeric characters 0 to 9. Some EBCDIC servers (Katakana and Hebrew character set A) do not use the standard representations of the lowercase alphabetic characters; use them with care when communicating with such servers.

The use of **NetName** is as follows:

1. If a name is supplied using the **NetName**, and it matches the name of an existing terminal resource in the server, the server attempts to reserve that terminal resource.
2. If a name is supplied, but it does not match the name of an existing terminal resource in the server, the server installs a terminal resource using the model terminal definition specified by the **DevType** parameter described below, and gives it the input name. (If **DevType** is a null pointer, CICS_EPI_ERR_TERMID_INVALID is returned for CICS_EPI_VERSION_200 or later, otherwise CICS_EPI_ERR_FAILED is returned.)
3. If **NetName** is a null pointer, a terminal resource is installed using the model terminal definition specified in **DevType**. If **DevType** is a null pointer, the selected terminal type is not predictable, so you are advised to use **DevType** to ensure consistent results. The name of the terminal resource is returned in the **NetName** field of the **CICS_EpiDetails_t** structure.

The EPI uses this parameter only for input.

DevType

A pointer to a null-terminated string that is used in the server to select a model terminal definition from which a terminal resource definition is generated, or a null pointer.

If a string is supplied that is shorter than CICS_EPI_DEVTYPE_MAX characters, it should be padded with nulls to a length of CICS_EPI_DEVTYPE_MAX + 1.

The string is transmitted to the server without conversion to uppercase.

The characters used are translated from the client's code page to an EBCDIC code page before transmission. If the server uses an ASCII code page, they will be retranslated. The only characters guaranteed to be invariant under these translations are the uppercase characters A

to Z, and the numeric characters 0 to 9. Some EBCDIC servers (Katakana and Hebrew character set A) do not use the standard representations of the lowercase alphabetic characters; use them with care when communicating with such servers.

The EPI uses this parameter only for input.

NotifyFn

A pointer to a callback routine that is called whenever an event occurs for the terminal resource, such as the arrival of an ATI request. If a callback routine is not required, this parameter should be set to null. Not supported in COBOL applications.

The EPI uses this parameter only for input.

Details

A pointer to the **CICS_EpiDetails_t** structure that on return contains various details about the terminal resource that was installed or reserved.

The EPI uses the fields in this structure only for output.

TermIndex

A pointer to a terminal index for the terminal resource just installed or reserved. The returned terminal index must be used as input to all further EPI function calls to identify the terminal resource to which the function is directed. The terminal index supplied is the first available integer starting from 0.

The EPI uses this parameter only for output.

Return codes

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_SYSTEM

The specified server is not known to the client.

CICS_EPI_ERR_SECURITY

The server rejected the attempt for security reasons.

CICS_EPI_ERR_NULL_PARM

TermIndex was a null pointer.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_ERR_SERVER_DOWN

The function failed because the server was down.

CICS_EPI_ERR_TERMID_INVALID

The function failed because an invalid TermId was supplied.

CICS_EPI_ERR_MODELID_INVALID

The function failed because an invalid Model terminal definition was supplied.

CICS_EPI_ERR_NOT_3270_DEVICE

The function failed because the device type supplied was not for a 3270 device.

CICS_EPI_ERR_ALREADY_INSTALLED

The function failed because the terminal was already installed.

CICS_EPI_ERR_SERVER_BUSY

The function failed because the server was busy.

CICS_EPI_ERR_RESOURCE_SHORTAGE

The CICS server or CICS Transaction Gateway did not have enough resources to complete the terminal install.

CICS_EPI_ERR_MAX_SESSIONS

The MAXREQUESTS limit has been exceeded.

CICS_EPI_ERR_MAX_SYSTEMS

An attempt was made to start connections to more servers than your configuration allows.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiAddExTerminal

The **CICS_EpiAddExTerminal** function installs a new terminal resource, or reserves an existing terminal resource, for the application's use.

CICS_EpiAddExTerminal	System
	NetName
	DevType
	NotifyFn
	Details
	TermIndex
	Attributes

Purpose

It provides a terminal index, which should be used to identify the terminal resource on all further EPI calls. It also provides the information defined in the **CICS_EpiDetails_t** data structure.

Some attributes, for example the character set and encoding scheme to be used for 3270 data and the sign-on capability, may be determined by the application. These attributes are specified in the **CCSID** and **SignonCapability** fields in the **CICS_EpiAttributes_t** structure.

Parameters

System

A pointer to a null-terminated string that specifies the name of the server in which the terminal resource is to be installed or reserved. If the name is shorter than **CICS_EPI_SYSTEM_MAX** characters, it must be padded with nulls to a length of **CICS_EPI_SYSTEM_MAX + 1**.

If the string is all nulls, the default server is selected by the EPI. To determine the name of the server chosen, use **CICS_EpiInquireSystem**.

The EPI uses this parameter only for input.

NetName

A pointer to a null-terminated string that specifies the name of the terminal resource to be installed or reserved, or null. The interpretation of this name is server-dependent.

If a string is supplied that is shorter than **CICS_EPI_NETNAME_MAX**, it must be padded with nulls to a length of **CICS_EPI_NETNAME_MAX + 1**.

The string is transmitted to the server without conversion to uppercase.

The characters used are translated from the client's code page to an EBCDIC code page before transmission. If the server uses an ASCII code page, they will be retranslated. The only characters guaranteed to be invariant under these translations are the uppercase characters A to Z, and the numeric characters 0 to 9. Some EBCDIC servers (Katakana and Hebrew character set A) do not use the standard representations of the lowercase alphabetic characters; use them with care when communicating with such servers.

The use of **NetName** is as follows:

1. If a name is supplied using the **NetName**, and it matches the name of an existing terminal resource in the server, the server attempts to reserve that terminal resource.
2. If a name is supplied, but does not match the name of an existing terminal resource in the server, the server installs a terminal resource using the model terminal definition specified by the **DevType** parameter described below, and gives it the input name. (If **DevType** is a null pointer, **CICS_EPI_ERR_TERMID_INVALID**

is returned for CICS_EPI_VERSION_200 or later, otherwise CICS_EPI_ERR_FAILED is returned.)

3. If **NetName** is a null pointer, a terminal resource is installed using the model terminal definition specified in **DevType**. If **DevType** is a null pointer, the selected terminal type is not predictable, so you are advised to use **DevType** to ensure consistent results. The name of the terminal resource is returned in the **NetName** field of the **CICS_EpiDetails_t** structure.

The EPI uses this parameter only for input.

DevType

A pointer to a null-terminated string that is used in the server to select a model terminal definition from which a terminal resource definition is generated, or a null pointer.

If a string is supplied that is shorter than CICS_EPI_DEVTYPE_MAX characters, it should be padded with nulls to a length of CICS_EPI_DEVTYPE_MAX + 1.

The string is transmitted to the server without conversion to uppercase.

The characters used are translated from the client's code page to an EBCDIC code page before transmission. If the server uses an ASCII code page, they will be retranslated. The only characters guaranteed to be invariant under these translations are the uppercase characters A to Z, and the numeric characters 0 to 9. Some EBCDIC servers (Katakana and Hebrew character set A) do not use the standard representations of the lowercase alphabetic characters; use them with care when communicating with such servers.

The EPI uses this parameter only for input.

NotifyFn

A pointer to a callback routine that is called whenever an event occurs for the terminal resource, such as the arrival of an ATI request. If a callback routine is not required, this parameter should be set to null. Not supported in COBOL applications.

The EPI uses this parameter only for input.

Details

A pointer to the **CICS_EpiDetails_t** structure that on return contains various details about the terminal resource that was installed or reserved. For asynchronous calls, the **Details** parameter should be set to NULL. If the pointer is not set to nulls, the details are added to the structure when the request to install the terminal resource has completed. For asynchronous calls this is done when the CICS_EPI_EVENT_ADD_TERM event occurs.

The EPI uses the fields in this structure only for output.

TermIndex

A pointer to a terminal index for the terminal resource just installed or reserved. The returned terminal index must be used as input to all further EPI function calls to identify the terminal resource to which the function is directed. The terminal index supplied is the first available integer starting from 0.

The EPI uses this parameter only for output.

Attributes

A pointer to the **CICS_EpiAttributes_t** structure that specifies attributes definable by the client application for the terminal resource that is to be installed *The structure must be set to nulls before use.*

Default attributes are assumed if the pointer is set to null.

The EPI uses this parameter only for input.

Return codes

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_SYSTEM

The specified server is not known to the CICS Transaction Gateway.

CICS_EPI_ERR_SECURITY

The server rejected the attempt for security reasons.

CICS_EPI_ERR_NULL_PARM

TermIndex was a null pointer.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_ERR_RESPONSE_TIMEOUT

No response was received from the server within the specified interval.

CICS_EPI_ERR_SIGNON_NOT_POSS

The server does not allow terminal resources to be installed as sign-on capable.

CICS_EPI_ERR_SERVER_DOWN

The function failed because the server was down.

CICS_EPI_ERR_PASSWORD_INVALID

The length of the password exceeds **CICS_EPI_PASSWORD_MAX**.

CICS_EPI_ERR_ADDTYPE_INVALID

The value assigned to the **EpiAddType** field in the **CICS_EpiAttributes_t** structure is neither **CICS_EPI_ADD_ASYNC** nor **CICS_EPI_ADD_SYNC**.

CICS_EPI_ERR_SIGNONCAP_INVALID

The value assigned to the **SignonCapability** field in the **CICS_EpiAttributes_t** structure is neither **CICS_EPI_SIGNON_CAPABLE** nor **CICS_EPI_SIGNON_INCAPABLE**.

CICS_EPI_ERR_USERID_INVALID

The length of the userid exceeds **CICS_EPI_USERID_MAX**.

CICS_EPI_ERR_TERMID_INVALID

The function failed because an invalid TermId was supplied.

CICS_EPI_ERR_MODELID_INVALID

The function failed because an invalid Model terminal definition was supplied.

CICS_EPI_ERR_NOT_3270_DEVICE

The function failed because the device type supplied was not for a 3270 device.

CICS_EPI_ERR_ALREADY_INSTALLED

The function failed because the terminal was already installed.

CICS_EPI_ERR_CCSID_INVALID

The function failed because an invalid CCSID was supplied.

For details on the CCSID values for various character sets, see *Data conversion when using the Client daemon*, in the *CICS Transaction Gateway: Administration* book for your operating system.

CICS_EPI_ERR_SERVER_BUSY

The function failed because the server was busy.

CICS_EPI_ERR_VERSION

The function is not supported for the version at which the EPI was initialized.

CICS_EPI_ERR_RESOURCE_SHORTAGE

The CICS server or CICS Transaction Gateway did not have enough resources to complete the terminal install.

CICS_EPI_ERR_MAX_SESSIONS

There were not enough communication resources to satisfy this request.

CICS_EPI_ERR_MAX_SYSTEMS

An attempt was made to start connections to more servers than your configuration allows.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiInquireSystem

CICS_EpiInquireSystem	TermIndex System
-----------------------	---------------------

Purpose

The **CICS_EpiInquireSystem** function returns the name of the server on which a given terminal resource (identified by its terminal index) is installed.

Parameters

TermIndex

The terminal index of the terminal resource whose location is to be determined.

The EPI uses this parameter only for input.

System

A pointer to a string of length CICS_ECI_SYSTEM_MAX + 1 in which the name of the server will be returned.

The EPI uses this parameter only for output.

Return codes

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_NULL_PARM

System was a null pointer.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NORMAL

The function completed successfully. The name of the server is returned in the **System** parameter padded with nulls to a length of CICS_EPI_SYSTEM_MAX + 1.

CICS_EpiDelTerminal

CICS_EpiDelTerminal	TermIndex
---------------------	-----------

Purpose

The **CICS_EpiDelTerminal** function deletes a previously added terminal resource. The application should not consider the deletion complete until it receives the corresponding CICS_EPI_EVENT_END_TERM event. The terminal index remains allocated until a **CICS_EpiGetEvent** call retrieves the CICS_EPI_EVENT_END_TERM event. A call to this function fails if the terminal resource is currently running a transaction. To ensure that a terminal resource is deleted, the application must wait until the current transaction finishes and process all outstanding events before issuing the **CICS_EpiDelTerminal** call.

If the terminal resource was autoinstalled, its definition is deleted from the server. When a **CICS_EpiDelTerminal** call has completed successfully for a terminal resource, use of the terminal index is restricted to **CICS_EpiGetEvent** calls until the application has received the corresponding CICS_EPI_EVENT_END_TERM event.

Parameters

TermIndex

The terminal index of the terminal resource to be deleted.

The EPI uses this parameter only for input.

Return codes

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_TRAN_ACTIVE

A transaction is currently running against the terminal resource, or there are unprocessed events for the terminal resource.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiPurgeTerminal

CICS_EpiPurgeTerminal

TermIndex

Purpose

The **CICS_EpiPurgeTerminal** function purges a previously added terminal resource. The application should not consider the deletion complete until it receives the corresponding CICS_EPI_EVENT_END_TERM event.

The **CICS_EpiPurgeTerminal** call differs from the **CICS_EpiDelTerminal** call in that the application does not have to wait until the current transaction finishes or process all outstanding events before issuing the call.

If the terminal resource was autoinstalled, its definition is deleted from the server.

This purge function does not cancel ATI requests already received by the server, and queued against the terminal.

Parameters

TermIndex

The terminal index of the terminal resource to be deleted.

The EPI uses this parameter only for input.

Return codes

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_ERR_VERSION

The function is not supported for the version at which the EPI was initialized.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiSetSecurity

CICS_EpiSetSecurity	TermIndex UserId Password
---------------------	---------------------------------

Purpose

The **CICS_EpiSetSecurity** function allows a client application to specify a userid and password to be associated with a terminal resource previously installed as sign-on incapable.

The **CICS_EpiSetSecurity** function may be invoked at any time; the userid and password will be used as further transactions are started for the terminal resource. A CICS Transaction Gateway determined userid and password will be used if the function either has not been invoked for the terminal resource or has been invoked and has set the userid, and by implication the password, to nulls.

Note that the client application is responsible for verifying the userid and password.

Parameters

TermIndex

The terminal index of the terminal.

The EPI uses this parameter only for input.

UserId

A pointer to a null-terminated string that specifies the userid. If the userid is shorter than CICS_EPI_USERID_MAX characters, it must be padded with nulls to a length of CICS_EPI_USERID_MAX+1.

The EPI uses this parameter only for input.

Password

A pointer to a null-terminated string that specifies the password. If the password is shorter than CICS_EPI_PASSWORD_MAX characters, it must be padded with nulls to a length of CICS_EPI_PASSWORD_MAX+1.

The EPI uses this parameter only for input.

Return codes

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_ERR_SYSTEM_ERROR

An internal system error occurred.

CICS_EPI_ERR_VERSION

The function is not supported for the version at which the EPI was initialized.

CICS_EPI_ERR_NULL_PASSWORD

Password was a null pointer.

CICS_EPI_ERR_NULL_USERID

Userid was a null pointer.

CICS_EPI_ERR_PASSWORD_INVALID

The length of the password exceeds CICS_EPI_PASSWORD_MAX.

CICS_EPI_ERR_USERID_INVALID

The length of the userid exceeds CICS_EPI_USERID_MAX.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiStartTran

Purpose

CICS_EpiStartTran	TermIndex
	TransId
	Data
	Size

The **CICS_EpiStartTran** function starts a new transaction from a terminal resource, or continues a pseudoconversation.

- *Starting a new transaction*—do this after **CICS_EpiAddTerminal**, or after a CICS_EPI_EVENT_END_TRAN event indicated that the previous transaction **did not** specify a transaction to process the next input from the terminal resource.
- *Continuing a pseudoconversation*—do this after a CICS_EPI_EVENT_END_TRAN event that indicated that the previous transaction specified **did** specify a transaction to process the next input from the terminal resource.

If the call is successful, no further start requests can be issued for this terminal resource until the transaction ends; this is indicated by the CICS_EPI_EVENT_END_TRAN event.

Parameters

TermIndex

The terminal index of the terminal resource that is to run the transaction.

The EPI uses this parameter only for input.

TransId

A pointer to a string specifying the transaction to be run, or the null pointer. If a new transaction is being started, and this input is the null pointer, the name of the transaction is extracted from the data stream supplied in the **Data** parameter. If a pseudoconversation is being continued, and the pointer is not null, the string must be the name of the transaction returned in the preceding CICS_EPI_EVENT_END_TRAN event for this terminal resource. If the pointer is not null, and the string is shorter than CICS_EPI_TRANSID_MAX characters, it should be padded with spaces to this length.

The EPI uses this parameter only for input.

Data

A pointer to the 3270 data stream to be associated with the transaction. This parameter must not be a null pointer, because the data stream must contain at least an AID byte.

If a new transaction is being started, and the **TransId** parameter is the null pointer, the data stream must be at least 4 bytes long, must contain the name of the transaction to be started, and might contain data to be supplied to the transaction on its first EXEC CICS RECEIVE command.

If a new transaction is being started, and the **TransId** parameter is not the null pointer, the data stream might be only one byte (an AID byte), or 3 bytes (an AID byte and a cursor address), or longer than 3 bytes (an AID byte, a cursor address, and data and SBA commands). In the last case, the data is supplied to the transaction program on the first EXEC CICS RECEIVE command.

If a pseudoconversation is being continued, the data stream might be only one byte (an AID byte), or 3 bytes (an AID byte and a cursor address), or longer than 3 bytes (an AID byte, a cursor address, and data and SBA commands). In the last case the data is supplied to the transaction program on the first EXEC CICS RECEIVE command.

The details of the format of 3270 data streams for CICS are described in *3270 data streams for the EPI*, in *CICS Transaction Gateway: Programming Guide*.

The length of the 3270 data stream must not exceed the value that was returned in **MaxData** in **CICS_EpiDetails_t** when the terminal resource was installed with **CICS_EpiAddTerminal**.

The EPI uses this parameter only for input.

Size The size in bytes of the initial data to be passed to the transaction.
The EPI uses this parameter only for input.

Note: The application might expect a terminal resource to be free to start a transaction and yet get an unexpected return code of **CICS_EPI_ERR_ATI_ACTIVE** from a call to **CICS_EpiStartTran**. If this happens, it means that the EPI has started an ATI request against the terminal resource and issued the corresponding **CICS_EPI_EVENT_START_ATI** event, but the application has not yet retrieved the event by issuing a **CICS_EpiGetEvent** call.

Return codes

CICS_EPI_ERR_ATI_ACTIVE

An ATI transaction is active for this terminal resource.

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NO_DATA

No initial data was supplied.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_TTI_ACTIVE

A transaction started from the EPI is already active for this terminal resource.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_ERR_SERVER_DOWN

The function failed because the server was down.

CICS_EPI_ERR_RESOURCE_SHORTAGE

The CICS server or CICS Transaction Gateway did not have enough resources to complete the terminal install.

CICS_EPI_ERR_MAX_SESSIONS

There were not enough communication resources to satisfy this request.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiReply

CICS_EpiReply	TermIndex
	Data
	Size

Purpose

The **CICS_EpiReply** function sends data from a terminal resource to a CICS transaction. It should only be issued in response to a **CICS_EPI_EVENT_CONVERSE** event.

Parameters

TermIndex

The terminal index of the terminal resource from which the data is being sent.

The EPI uses this parameter only for input.

Data A pointer to the 3270 data stream to be sent to the transaction. This parameter must not be a null pointer, because the data stream must contain at least an AID byte. The data stream might be one byte (an AID byte), 3 bytes (an AID byte and a cursor address), or more than 3 bytes (an AID byte, a cursor address, and data and SBA commands). In the last case, what follows the cursor address is supplied to the transaction program on the first EXEC CICS RECEIVE command.

The length of the 3270 data stream must not exceed the value that was returned in **MaxData** in **CICS_EpiDetails_t** when the terminal resource was installed with **CICS_EpiAddTerminal**.

The EPI uses this parameter only for input.

Size The size of the data in bytes.

The EPI uses this parameter only for input.

Return codes

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NO_CONVERSE

No reply is expected by the terminal resource.

CICS_EPI_ERR_NO_DATA

No reply data was supplied.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_ERR_SERVER_DOWN

The function failed because the server was down.

CICS_EPI_ERR_ABENDED

The read timeout period has expired and the conversation has abended, but the **CICS_EPI_EVENT_END_TRAN** event has not yet been received by the application.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiATISState

CICS_EpiATISState	TermIndex ATISState
--------------------------	--------------------------------------

Purpose

The **CICS_EpiATISState** function allows the calling application to query and alter the way in which ATI requests for a terminal resource are handled. If ATI requests are enabled (**CICS_EPI_ATI_ON**) and an ATI request is issued in the server, the request is started when the terminal resource becomes free. If ATI requests are held (**CICS_EPI_ATI_HOLD**), any ATI requests issued are queued, and started when ATI requests are next enabled.

The state for ATI requests after a **CICS_EpiAddTerminal** call is **CICS_EPI_ATI_HOLD**. The EPI application may change the state to **CICS_EPI_ATI_ON** when it is ready to allow ATI requests to be processed. (The server also maintains a ATI state for terminal resources, which is independent of the ATI state maintained in the EPI. Changes to the ATI state on the server do not affect the ATI status in the EPI.)

Parameters

TermIndex

The terminal index of the terminal resource whose ATI state is required.

The EPI uses this parameter only for input.

ATISState

The EPI uses this parameter for both input and output depending on the input value as follows:

CICS_EPI_ATI_ON

Enable ATI requests, and return the previous ATI state in this parameter.

CICS_EPI_ATI_HOLD

Hold ATI requests until they are next enabled, and return the previous ATI state in this parameter.

CICS_EPI_ATI_QUERY

Do not change the ATI state; just return the current state in this parameter.

Return codes

CICS_EPI_ERR_ATI_STATE

An invalid **ATISState** value was provided.

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NULL_PARAM

ATISState was a null pointer.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiGetEvent

CICS_EpiGetEvent	TermIndex
	Wait
	Event

Purpose

The **CICS_EpiGetEvent** function obtains information about an event that has occurred for a terminal resource.

Remember that this call may be attempted only from the application, not from the callback routine.

Parameters

TermIndex

The terminal index of the terminal resource for which to obtain an event. This can be set to the constant **CICS_EPI_TERM_INDEX_NONE** to indicate that the next event for any terminal resource used by this application is to be returned. The application can examine the **TermIndex** field in the returned **CICS_EpiEventData_t** structure to determine the terminal resource against which the event was generated.

The EPI uses this parameter for both input and output.

Wait An indication of what should happen if no event has been generated for the terminal resource. Use one of the following values:

CICS_EPI_WAIT

Do not return until the next event occurs.

CICS_EPI_NOWAIT

Return immediately with an error code. This option is used if the application elects to poll for events.

The EPI uses this parameter only for input.

Event A pointer to a **CICS_EpiEventData_t** structure that on return contains the details of the event that occurred. The **Data** field in the structure should be set to point to the data buffer that is updated with any terminal data stream associated with the event. The **Size** field should be set to indicate the maximum size of this buffer, and is updated to contain the actual length of data returned.

Return codes

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_MORE_DATA

The supplied data buffer was not large enough to contain the terminal data; the data has been truncated.

CICS_EPI_ERR_MORE_EVENTS

An event was successfully obtained, but there are more events outstanding against this terminal resource.

CICS_EPI_ERR_NO_EVENT

No events are outstanding for this terminal resource.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_WAIT

The **Wait** parameter is not valid.

CICS_EPI_ERR_NULL_PARM

Event is a null pointer.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NORMAL

The function completed successfully, and there are no more events.

EPI events

EPI events occur when CICS has data to pass to the EPI application. The application can handle EPI events in a variety of ways. See *Events and callbacks*, in *CICS Transaction Gateway: Programming Guide*. Whichever mechanism is used, the data from CICS is obtained by calling **CICS_EpiGetEvent**.

CICS_EPI_EVENT_ADD_TERM

Purpose

The **CICS_EPI_EVENT_ADD_TERM** event indicates that an asynchronous request to install a terminal resource has completed. If the terminal resource was installed details will have been placed in the **CICS_EpiDetails_t** structure, pointed to by **Data**.

Fields completed

Event The **CICS_EPI_EVENT_ADD_TERM** event code.

EndReturnCode

The reason for termination. Refer to the **CICS_EpiAddExTerminal** function for details of return codes.

Data A pointer to the **CICS_EpiDetails_t** structure that is updated with the terminal details, if the **EndReturnCode** is **CICS_EPI_NORMAL**.

CICS_EPI_EVENT_SEND

Purpose

The **CICS_EPI_EVENT_SEND** event indicates that a transaction has sent some 3270 data to a terminal resource, typically as a result of an EXEC CICS SEND command. No reply is expected, and none should be attempted.

Fields completed

Event The **CICS_EPI_EVENT_SEND** event code.

Data A pointer to the buffer that is updated to contain the data sent by the transaction. See *3270 data streams for the EPI*, in *CICS Transaction Gateway: Programming Guide*, for details of the data stream format.

Size The length of the data in the **Data** buffer.

CICS_EPI_EVENT_CONVERSE

Purpose

The **CICS_EPI_EVENT_CONVERSE** event indicates that a transaction is expecting a reply as a result of either an EXEC CICS RECEIVE command, or an EXEC CICS CONVERSE command.

The application should issue a **CICS_EpiReply** call to return the data to CICS, as follows:

- If the transaction has issued an EXEC CICS RECEIVE command without specifying the **BUFFER** option, the buffer might contain data sent from the transaction, or it might be empty. If there is data to process, deal with it before replying. Send the reply when the data to be sent is available.
- If the transaction has issued an EXEC CICS RECEIVE BUFFER command, the data buffer contains the 3270 Read Buffer command and the **Size** field is set to 1. The reply should be sent immediately.

Fields completed

Event The **CICS_EPI_EVENT_CONVERSE** event code.

Data A pointer to the buffer that is updated to contain the data sent by the transaction, as defined above.

Size The length of the data in the buffer. This may be set to zero to indicate that no data was sent, but a reply is still expected.

CICS_EPI_EVENT_END_TRAN

Purpose

The CICS_EPI_EVENT_END_TRAN event indicates the end of a transaction that was running against a terminal resource. If the transaction failed, the **EndReason** and **EndReturnCode** specify the cause. If the transaction completed normally, the **EndReason** field is set to CICS_EPI_TRAN_NO_ERROR and **EndReturnCode** is set to CICS_EPI_NORMAL. If the transaction was pseudoconversational, the **TransId** field contains the name of the next transaction required. The application should start this transaction by issuing a **CICS_EpiStartTran** call.

The CICS_EPI_EVENT_END_TRAN event occurs when a transaction running against a terminal resource abends or ends following execution of a RETURN command for which the IMMEDIATE option was not specified.

Fields completed

Event The CICS_EPI_EVENT_END_TRAN event code.

EndReason

An indication of what caused the end transaction event. It can be one of the following values:

CICS_EPI_TRAN_NO_ERROR

Normal transaction termination.

CICS_EPI_TRAN_NOT_STARTED

The transaction failed to start.

CICS_EPI_TRAN_STATE_UNKNOWN

The transaction failed to complete.

CICS_EPI_READTIMEOUT_EXPIRED

The read timeout expired.

TransId

The name of the next transaction to start, if the previous transaction was pseudoconversational. This name is 4 characters long and null-terminated. If there is no next transaction, the field is set to nulls.

EndReturnCode

A string containing the CICS_EPI_returncode.

CICS_EPI_EVENT_START_ATI

Purpose

The CICS_EPI_EVENT_START_ATI event indicates that an ATI transaction has been started against the terminal resource. If the terminal resource receives an ATI request while it is running another transaction, the request is held until the transaction ends. The transaction is then started on behalf of the terminal resource, and the CICS_EPI_EVENT_START_ATI event is generated to inform the application.

Fields completed

Event The CICS_EPI_EVENT_START_ATI event code.

TransId

The name of the transaction that was started. This name is 4 characters long and null-terminated.

CICS_EPI_EVENT_END_TERM

Purpose

The CICS_EPI_EVENT_END_TERM event indicates that a terminal resource no longer exists. After this event, the terminal index that was previously used for the terminal resource is not valid. If the EPI detects that a CICS server has shut down, CICS_EPI_EVENT_END_TERM events are generated for all terminal resources that the application has installed in that server and not subsequently deleted.

Fields completed

Event The CICS_EPI_EVENT_END_TERM event code.

EndReason

An indication of why the terminal resource was deleted. It can be one of the following values:

CICS_EPI_END_SIGNOFF

The terminal resource was signed off. This can be as a result of running the CESF transaction or of calling the **CICS_EpiDelTerminal** function.

CICS_EPI_END_SHUTDOWN

The CICS server is shutting down.

CICS_EPI_END_OUTSERVICE

The terminal resource has been switched out of service.

CICS_EPI_END_UNKNOWN

An unexpected error has occurred.

CICS_EPI_END_FAILED

An attempt to delete a terminal resource failed.

External Security Interface

ESI constants and data structures

This section describes the constants and data structures that you need to use the ESI.

ESI constants

The following constants are referred to symbolically in the descriptions of the ESI data structures, and functions in this information. Their values are given here to help you understand the descriptions. However, your code should always use the symbolic names of ESI constants provided for the programming language you are using.

Lengths of fields

- CICS_ESI_PASSWORD_MAX (10)
- CICS_ESI_SYSTEM_MAX (8)
- CICS_ESI_USERID_MAX (10)

ESI data structures

The following data structures are available for use with the ESI.

- CICS_EsiDate_t
- CICS_EsiTime_t
- CICS_EsiDetails_t

In the descriptions of the fields in the data structures, fields described as strings are null-terminated strings.

CICS_EsiDate_t:

Purpose

The **CICS_EsiDate_t** structure contains a date represented as year, month, and day.

Fields

Year 4-digit year held in **cics_ushort_t** format.

Month

Month held in **cics_ushort_t** format; values range from 1 to 12 with 1 representing January.

Day

Day held in **cics_ushort_t** format; values range from 1 to 31 with 1 representing the first day of the month.

CICS_EsiTime_t:**Purpose**

The **CICS_EsiTime** structure contains a time represented as hours, minutes, seconds, and hundredths of a second.

Fields

Hours Hours held in **cics_ushort_t** format; values range from 0 to 23.

Minutes

Minutes held in **cics_ushort_t** format; values range from 0 to 59.

Seconds

Seconds held in **cics_ushort_t** format; values range from 0 to 59.

Hundredths

Hundredths of a second held in **cics_ushort_t** format; values range from 0 to 99.

CICS_EsiDetails_t:**Purpose**

The **CICS_EsiDetails_t** structure contains information returned from a successful invocation of either the **CICS_VerifyPassword** or the **CICS_ChangePassword** functions.

Fields**LastVerifiedDate**

The date on which the password was last verified.

LastVerifiedTime

The time at which the password was last verified.

ExpiryDate

The date on which the password will expire.

ExpiryTime

The time at which the password will expire.

LastAccessDate

The date on which the userid was last accessed.

LastAccessTime

The time at which the userid was last accessed.

InvalidCount

The number of times that an invalid password has been entered for the userid.

ESI functions

This section describes the functions provided by the ESI that can be called from an application program:

- **CICS_VerifyPassword**
- **CICS_ChangePassword**
- **CICS_SetDefaultSecurity**

CICS_VerifyPassword

CICS_VerifyPassword	UserId
	Password
	System
	Details

Purpose

The **CICS_VerifyPassword** function allows a client application to verify that a password matches the password recorded by an external security manager for a specified userid.

Note that the external security manager is assumed to be located in a server to which the client is connected.

Parameters**UserId**

A pointer to a null-terminated string that specifies the userid whose password is to be verified. If the userid is shorter than CICS_ESI_USERID_MAX characters, it must be padded with nulls to a length of CICS_ESI_USERID_MAX+1.

The ESI uses this parameter only for input.

Password

A pointer to a null-terminated string that specifies the password to be checked by the external security manager for the specified userid. If

the password is shorter than `CICS_ESI_PASSWORD_MAX` characters, it must be padded with nulls to a length of `CICS_ESI_PASSWORD_MAX+1`.

The ESI uses this parameter only for input.

System

A pointer to a null-terminated string that specifies the name of the server in which the password is to be verified. If the name is shorter than `CICS_ESI_SYSTEM_MAX` characters, it must be padded with nulls to a length of `CICS_ESI_SYSTEM_MAX+1`.

If the string is all nulls, the default server is selected.

The ESI uses this parameter only for input.

Details

A pointer to the `CICS_EsiDetails_t` structure that on return contains further information returned by the external security manager.

The ESI uses the fields in this structure only for output.

Return codes

CICS_ESI_NO_ERROR

The function completed successfully.

CICS_ESI_ERR_CALL_FROM_CALLBACK

The function was invoked from a callback routine.

CICS_ESI_ERR_SYSTEM_ERROR

An internal system error occurred.

CICS_ESI_ERR_NO_CICS

The CICS Transaction Gateway is unavailable, or the specified server is unavailable.

CICS_ESI_ERR_CICS_DIED

The specified server is no longer available.

CICS_ESI_ERR_RESOURCE_SHORTAGE

The CICS Transaction Gateway did not have enough resources to complete the request.

CICS_ESI_ERR_NO_SESSIONS

The application has as many outstanding ECI and EPI requests as the configuration will support.

CICS_ESI_ERR_UNKNOWN_SERVER

The requested server could not be located. Only servers returned by the `CICS_EciListSystems` and `CICS_EpiListSystems` functions are acceptable.

CICS_ESI_ERR_MAX_SESSIONS

There were not enough communications resources to satisfy the request. Consult the documentation for your CICS Transaction Gateway or server to see how to control the number of servers you can use.

CICS_ESI_ERR_MAX_SYSTEMS

You tried to start requests to more servers than your configuration allows. Consult the documentation for your CICS Transaction Gateway or server to see how to control the number of servers you can use.

CICS_ESI_ERR_NULL_USERID

The userid is set to nulls.

CICS_ESI_ERR_NULL_PASSWORD

The password is set to nulls.

CICS_ESI_ERR_PEM_NOT_SUPPORTED

Password expiry management is supported only for communications with the requested server over SNA.

CICS_ESI_ERR_PEM_NOT_ACTIVE

The requested server does not support password expiry management.

CICS_ESI_ERR_PASSWORD_EXPIRED

The password has expired.

CICS_ESI_ERR_PASSWORD_INVALID

The password is invalid.

CICS_ESI_ERR_USERID_INVALID

The userid is not known to the external security manager.

CICS_ESI_ERR_SECURITY_ERROR

An error has been detected by the external security manager. The most likely explanation is that the userid has been revoked.

The mapping of actual return code values to the symbolic names is contained in the <install_path>\include\cics_esi.h file. COBOL users can find it in the <install_path>\copybook\cicsesi.cbl file.

CICS_ChangePassword

CICS_ChangePassword	UserId
	OldPassword
	NewPassword
	System
	Details

Purpose

The **CICS_ChangePassword** function allows a client application to change the password recorded by an external security manager for a specified userid.

Note that the external security manager is assumed to be located in a server to which the CICS Transaction Gateway is connected.

Parameters

UserId

A pointer to a null-terminated string that specifies the userid whose password is to be changed. If the userid is shorter than CICS_ESI_USERID_MAX characters, it must be padded with nulls to a length of CICS_ESI_USERID_MAX+1.

The ESI uses this parameter only for input.

OldPassword

A pointer to a null-terminated string that specifies the current password for the specified userid. If the password is shorter than CICS_ESI_PASSWORD_MAX characters, it must be padded with nulls to a length of CICS_ESI_PASSWORD_MAX+1.

The ESI uses this parameter only for input.

NewPassword

A pointer to a null-terminated string that specifies the new password for the specified userid. If the password is shorter than CICS_ESI_PASSWORD_MAX characters, it must be padded with nulls to a length of CICS_ESI_PASSWORD_MAX+1.

The password is changed only if the currently password is correctly specified.

The ESI uses this parameter only for input.

System

A pointer to a null-terminated string that specifies the name of the server in which the password is to be verified. If the name is shorter than CICS_ESI_SYSTEM_MAX characters, it must be padded with nulls to a length of CICS_ESI_SYSTEM_MAX+1.

If the string is all nulls, the default server is selected.

The ESI uses this parameter only for input.

Details

A pointer to the **CICS_EsiDetails_t** structure that on return contains further information returned by the external security manager.

The ESI uses the fields in this structure only for output.

Return codes

CICS_ESI_NO_ERROR

The function completed successfully.

CICS_ESI_ERR_CALL_FROM_CALLBACK

The function was invoked from a callback routine.

CICS_ESI_ERR_SYSTEM_ERROR

An internal system error occurred.

CICS_ESI_ERR_NO_CICS

The CICS Transaction Gateway is unavailable, or the specified server is unavailable.

CICS_ESI_ERR_CICS_DIED

The specified server is no longer available. To confirm that the password has been changed, use the **CICS_VerifyPassword** function.

CICS_ESI_ERR_RESOURCE_SHORTAGE

The CICS Transaction Gateway did not have enough resources to complete the request.

CICS_ESI_ERR_NO_SESSIONS

The application has as many outstanding ECI and EPI requests as the configuration will support.

CICS_ESI_ERR_UNKNOWN_SERVER

The requested server could not be located. Only servers returned by the **CICS_EciListSystems** and **CICS_EpiListSystems** functions are acceptable.

CICS_ESI_ERR_MAX_SESSIONS

There were not enough communications resources to satisfy the request. Consult the documentation for your CICS Transaction Gateway or server to see how to control the number of servers you can use.

CICS_ESI_ERR_MAX_SYSTEMS

You tried to start requests to more servers than your configuration allows. Consult the documentation for your CICS Transaction Gateway or server to see how to control the number of servers you can use.

CICS_ESI_ERR_NULL_USERID

The userid is set to nulls.

CICS_ESI_ERR_NULL_OLD_PASSWORD

The current password is set to nulls.

CICS_ESI_ERR_NULL_NEW_PASSWORD

The new password is set to nulls.

CICS_ESI_ERR_PEM_NOT_SUPPORTED

Password expiry management is supported only for communications with the requested server over SNA.

CICS_ESI_ERR_PEM_NOT_ACTIVE

The requested server does not support password expiry management.

CICS_ESI_ERR_PASSWORD_INVALID

The password is invalid.

CICS_ESI_ERR_PASSWORD_REJECTED

The new password does not confirm to the standards defined for the external security manager.

CICS_ESI_ERR_USERID_INVALID

The userid is not known to the external security manager.

CICS_ESI_ERR_SECURITY_ERROR

An error has been detected by the external security manager. The most likely explanation is that the userid has been revoked.

The mapping of actual return code values to the symbolic names is contained in the <install_path>\include\cics_esi.h file. COBOL users can find it in the <install_path>\copybook\cicsesi.cbl file.

CICS_SetDefaultSecurity

CICS_SetDefaultSecurity	UserId
	Password
	System

Purpose

The **CICS_SetDefaultSecurity** function allows a client application to specify a default userid and password to be used for ECI and EPI requests passed to the server.

The userid, and the password, can be set to nulls, that is, binary zeroes. In this case the default userid and password are unset, so that CICS Transaction Gateway acts as if no userid and password has been set.

The userid, and the password, can also be set to spaces. However, this is valid only if **Usedfltuser=yes** is specified in the CICS connection definition. In this

case CICS uses its default userid. Refer to the documentation for your CICS server for more information on the Usedfltuser specification.

The client application is responsible for verifying the userid and password.

Note that the userid and password, if required, may be obtained from any one of several places. The assumption is that the CICS Transaction Gateway uses the following search order:

1. Either the ECI parameter block for the ECI or the terminal specific values set by the **CICS_EpiSetSecurity** function.
2. The server specific values set by the **CICS_SetDefaultSecurity** function.
3. Defaults, for example the Windows userid, from the CICS Transaction Gateway's pop up window, and so on

Parameters

UserId

A pointer to a null-terminated string that specifies the userid to be set. If the userid is shorter than CICS_ESI_USERID_MAX characters, it must be padded with nulls to a length of CICS_ESI_USERID_MAX+1.

The ESI uses this parameter only for input.

Password

A pointer to a null-terminated string that specifies the password to be set for the specified userid. If the password is shorter than CICS_ESI_PASSWORD_MAX characters, it must be padded with nulls to a length of CICS_ESI_PASSWORD_MAX+1.

The ESI uses this parameter only for input.

System

A pointer to a null-terminated string that specifies the name of the server for which the password and userid are to be set. If the name is shorter than CICS_ESI_SYSTEM_MAX characters, it must be padded with nulls to a length of CICS_ESI_SYSTEM_MAX+1.

If the string is all nulls, the default server is selected.

The ESI uses this parameter only for input.

Return codes

CICS_ESI_NO_ERROR

The function completed successfully.

CICS_ESI_ERR_CALL_FROM_CALLBACK

The function was invoked from a callback routine.

CICS_ESI_ERR_SYSTEM_ERROR

An internal system error occurred.

CICS_ESI_ERR_NO_CICS

The CICS Transaction Gateway is unavailable, or the specified server is unavailable.

CICS_ESI_ERR_UNKNOWN_SERVER

The requested server could not be located. Only servers returned by the **CICS_EciListSystems** and **CICS_EpiListSystems** functions are acceptable.

CICS_ESI_ERR_USERID_INVALID

The length of the userid exceeds CICS_ESI_USERID_MAX.

CICS_ESI_ERR_PASSWORD_INVALID

The length of the password exceeds CICS_ESI_PASSWORD_MAX.

The mapping of actual return code values to the symbolic names is contained in the <install_path>\include\cics_esi.h file. COBOL users can find it in the <install_path>\copybook\cicsesi.cbl file.

Chapter 5. Java request monitoring, C ECI and C EPI exits

This information contains reference material for the Java request monitoring, C ECI and C EPI user exits.

For information about installing and using the exits, see *CICS Transaction Gateway: Programming Guide*.

The C exit routines are described under the following headings:

- **Purpose**—describes the kind of processing that the exit is intended to perform.
- **When called**—describes where in ECI or EPI processing the exit is called.
- **Parameters**—describes the parameters supplied to the exit. Parameters are classified as follows:
 - **Input**—the exit may look at it, but must not change it.
 - **Output**—the exit must not look at it, but must store a value in it.
 - **Input-output**—the exit may look at it, and may store a value in it.
- **Return codes**—describes the possible values the exit can return to the ECI or EPI. In each case the subsequent behavior of the ECI or EPI is described.

Java request monitoring exits

Online programming reference information is provided for the Java classes and interfaces provided with CICS Transaction Gateway.

The reference information is in HTML format and is generated using the Javadoc tool provided with the JDK.

Java request monitoring exits are only available in the CICS Transaction Gateway.

See the README file for the latest information on using the programming reference information.

Related information

Request monitoring user exit API information

C ECI exits reference

In this section the following exits are discussed:

- **CICS_EciInitializeExit**
- **CICS_EciTerminateExit**
- **CICS_EciExternalCallExit1**
- **CICS_EciExternalCallExit2**
- **CICS_EciSystemIdExit**
- **CICS_EciDataSendExit**
- **CICS_EciDataReturnExit**
- **CICS_EciSetProgramAliasExit**

Table 2 summarizes the exit names, the parameters passed to each exit, and the possible return codes.

Table 2. Summary of ECI exits

Function name	Parameters	Return codes:
CICS_EciInitializeExit	Version Anchor	CICS_EXIT_OK CICS_EXIT_NO_EXIT CICS_EXIT_CANT_INIT_EXITS user-defined
CICS_EciTerminateExit	Anchor	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_STORAGE user-defined
CICS_EciExternalCallExit1	Anchor Token ParmPtr	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
CICS_EciExternalCallExit2	Anchor Token ParmPtr	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
CICS_EciSystemIdExit	Anchor Token ParmPtr Reason	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM CICS_EXIT_GIVE_UP user_defined

Table 2. Summary of ECI exits (continued)

Function name	Parameters	Return codes:
CICS_EciDataSendExit	Anchor Token	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined
CICS_EciDataReturnExit	Anchor Token ParmPtr	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined
CICS_EciSetProgramAliasExit	Anchor EciParms Program	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined

Identification token

In order for the exits to be able to relate calls for the same ECI request, an *identification token* is passed in as a parameter to all exits except **CICS_EciInitializeExit** and **CICS_EciTerminateExit**. The token is the same for **CICS_EciExternalCallExit1** and **CICS_EciExternalCallExit2** that relate to the same call, and on intervening **CICS_EciDataSendExit**, **CICS_EciDataReturnExit**, and **CICS_EciSystemIdExit** exits. (Note that **CICS_EciExternalCallExit1** and **CICS_EciExternalCallExit2** are not called for a reply solicitation request.)

The token is unique within the operating system that initiated the request, for the duration of the request. It may be reused once the last exit for the request has been called.

In the case of an extended logical unit of work, the token may be different on different requests within the logical unit of work. (Since we allow reuse of the token, and a new program link call may not be made until the ECI_GET_REPLY request for the previous asynchronous request has completed, it may also be the same.)

The token is 8 bytes long. 8 null bytes is not a valid value for the token and is not supplied to the exits.

Process model implementation

All exits that relate to a particular request (i.e. have the same identification token) are called in the context of the application process.

CICS_EciInitializeExit

Function name:	Parameters
CICS_EciInitializeExit	Version Anchor

Purpose

To allow the user to set up an exit environment.

When called

On the first invocation of **CICS_ExternalCall**, for each process, after parameter validation has occurred.

Parameters

Version

Input parameter. The version of the ECI under which the exit is running.

Anchor

Output parameter. A pointer to a pointer that will be passed to the ECI exits. The second pointer is not used by the ECI; it is passed to the exits as supplied. You can acquire storage in this exit and pass its address to the other exits.

Return codes

CICS_EXIT_OK

The ECI continues processing this request, calling the exits where appropriate.

CICS_EXIT_NO_EXIT

The ECI continues processing this request, but does not call any more exits.

CICS_EXIT_CANT_INIT_EXITS

The ECI writes a CICS Transaction Gateway trace record, and then continues processing this request, but does not call any more exits.

user-defined

User-defined return codes must have a value not less than **CICS_EXIT_USER_BASE**. The ECI writes a CICS Transaction Gateway trace record, and then continues processing this request, but does not call any more exits.

CICS_EciTerminateExit

Function name:	Parameters
CICS_EciTerminateExit	Anchor

Purpose

To allow the user to clean up the exit environment. Any storage acquired by **CICS_EciInitializeExit** must be released in this exit.

CICS_EciTerminateExit is not called by the Client daemon.

When called

On termination of the process that issued the **CICS_EciInitializeExit**.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EciInitializeExit**.

Return codes

CICS_EXIT_OK

Termination continues.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The ECI writes a CICS Transaction Gateway trace record, and then continues with termination.

CICS_EXIT_BAD_STORAGE

CICS detected a storage error. The ECI writes a CICS Transaction Gateway trace record, and then continues with termination.

user-defined

User-defined return codes must have a value not less than **CICS_EXIT_USER_BASE**. The ECI writes a CICS Transaction Gateway trace record, and then continues with termination.

CICS_EciExternalCallExit1

Function name: CICS_EciExternalCallExit1	Parameters: Anchor Token ParmPtr
--	--

Purpose

To allow the user to pick the best system to run the program. This exit is called exactly once on each program link and each status information call. It is not called on a reply solicitation call. Although the exit is called when **eci_luw_token** is not zero, any change it makes to **eci_system_name** is ignored, as the server was selected when the logical unit of work was started.

When called

On invocation of **CICS_ExternalCall**, for each program link call and each status information call, after the ECI has validated the parameters.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EciInitializeExit**.

Token Input parameter. The identification token established by the ECI for this request.

ParmPtr

Input parameter. A pointer to the ECI parameter block. The exit must treat all fields in the ECI parameter block as inputs, except the **eci_system_name** field, which it may change.

Return codes

CICS_EXIT_OK

The ECI continues to process the request with the **eci_system_name** now specified in the ECI parameter block.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The ECI writes a CICS Transaction Gateway trace record, and then continues to process the request with the **eci_system_name** now specified in the ECI parameter block.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The ECI writes a CICS

Transaction Gateway trace record, and then continues to process the request with the **eci_system_name** now specified in the ECI parameter block.

user-defined

User-defined return codes must have a value not less than CICS_EXIT_USER_BASE. The ECI writes a CICS Transaction Gateway trace record, and then continues to process the request with the **eci_system_name** now specified in the ECI parameter block.

Notes

There is a limited set of conditions under which the exit may select a new system. The exit may select a system if the call is a program link or status information call, and if a new logical unit of work is being started. In other cases, the exit should return CICS_EXIT_OK.

If the calling application has put binary zeros as the system name in the parameter block, then the application is expecting that the system will be dynamically selected, and the exit may safely select the system.

If however the calling application has placed a system name in the parameter block, or if the application is a version 0 application, then it may not be expecting the target system to change, and application errors could result. In this case the exit would generally return without specifying a replacement system, with the result that the specified or default system name is to be used. If the exit chooses to change the selected system in this situation, then it may do so, but the following should be borne in mind.

- The exit routine must be sensitive to whether or not the modification of the target system will cause errors in the ECI application running on the client.
- The exit routine must maintain a knowledge base, keyed on appropriate data available to it, to enable it to determine whether this modification is acceptable to the client application.

CICS_EciExternalCallExit2

Function name: CICS_EciExternalCallExit2	Parameters: Anchor Token ParmPtr
--	--

Purpose

To allow the user to see the results of synchronous ECI calls for information gathering purposes only. This exit is called exactly once on every application

program link or status information call. It is not called on reply solicitation calls.

When called

Before the ECI call returns to the application, and after the return data is filled into the ECI parameter block.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EciInitializeExit**.

Token Input parameter. The identification token established by the ECI for this request.

ParmPtr

Input parameter. A pointer to the ECI parameter block. The exit must treat all fields in the ECI parameter block as inputs.

Return codes

CICS_EXIT_OK

The ECI returns control to the application that issued the **CICS_ExternalCall** request.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The ECI writes a CICS Transaction Gateway trace record, and then returns control to the application that issued the **CICS_ExternalCall** request.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The ECI writes a CICS Transaction Gateway trace record, and then returns control to the application that issued the **CICS_ExternalCall** request.

user-defined

User-defined return codes must have a value not less than **CICS_EXIT_USER_BASE**. The ECI writes a CICS Transaction Gateway trace record, and then returns control to the application that issued the **CICS_ExternalCall** request.

CICS_EciSystemIdExit

Function name: CICS_EciSystemIdExit	Parameters: Anchor Token ParmPtr Reason
---	--

Purpose

To allow the user to supply a new system name when the name supplied in the ECI parameter block is not valid.

When called

This exit is called when an error occurs that may be corrected by selection of a new system, userid, or password. This would be when the ECI has returned one of the following codes:

- ECI_ERR_NO_CICS
- ECI_ERR_UNKNOWN_SERVER
- ECI_ERR_SECURITY_ERROR
- ECI_ERR_SYSTEM_ERROR
- ECI_ERR_RESOURCE_SHORTAGE
- ECI_ERR_MAX_SYSTEMS.

It may be called when either when the Client daemon detects an error before data is sent to the server, or after data returns from the server.

Parameters

Anchor

Input parameter. The pointer set up by `CICS_EciInitializeExit`.

Token Input parameter. The identification token established by the ECI for this request.

ParmPtr

Input parameter. A pointer to the ECI parameter block. The exit must treat all fields in the ECI parameter block as inputs, except the following, which it may set:

- `eci_system_name`
- `eci_userid`
- `eci_password`.

Reason

Input parameter. A standard ECI error code that explains why the application request has not so far succeeded.

Return codes

CICS_EXIT_OK

The ECI retries the application call using the new parameters in the

ECI parameter block. (The CICS program communication area supplied by the application to the **CICS_ExternalCall** is preserved.) The application callback routine will not be called, nor will **CICS_EciExternalCallExit2**.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The ECI writes a CICS Transaction Gateway trace record, and then returns to the application that issued the **CICS_ExternalCall** request.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The ECI writes a CICS Transaction Gateway trace record, and then returns to the application that issued the **CICS_ExternalCall** request.

CICS_EXIT_GIVE_UP

The ECI returns to the application that issued the **CICS_ExternalCall** request.

user-defined

User-defined return codes must have a value not less than **CICS_EXIT_USER_BASE**. The ECI writes a CICS Transaction Gateway trace record, and then retries the application call as described for **CICS_EXIT_OK**.

CICS_EciDataSendExit

Function name: CICS_EciDataSendExit	Parameters: Anchor Token
---	---------------------------------------

Purpose

To allow the user to time calls for performance analysis.

When called

As close as possible to the time that the request will be sent to the server.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EciInitializeExit**.

Token Input parameter. The identification token established by the ECI for this request.

Return codes

CICS_EXIT_OK

The ECI continues processing the request.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The ECI writes a CICS Transaction Gateway trace record, and then continues processing the request.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The ECI writes a CICS Transaction Gateway trace record, and then continues processing the request.

user-defined

User-defined return codes must have a value not less than CICS_EXIT_USER_BASE. The ECI writes a CICS Transaction Gateway trace record, and then continues processing the request.

CICS_EciDataReturnExit

Function name: CICS_EciDataReturnExit	Parameters: Anchor Token ParmPtr
---	--

Purpose

To allow the user to time calls for performance analysis.

When called

As close as possible to the time that the response from the server has been received, and the ECI block and commarea data for eventual return to the application has been built. It is also called if there is a timeout because of a lack of response from the server.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EciInitializeExit**.

Token Input parameter. The identification token established by the ECI for this request.

ParmPtr

Input parameter. A pointer to the ECI parameter block. The exit must treat all fields in the ECI parameter block as inputs.

Return codes**CICS_EXIT_OK**

The ECI continues processing the request.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The ECI writes a CICS Transaction Gateway trace record, and then continues processing the request.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The ECI writes a CICS Transaction Gateway trace record, and then continues processing the request.

user-defined

User-defined return codes must have a value not less than CICS_EXIT_USER_BASE. The ECI writes a CICS Transaction Gateway trace record, and then continues processing the request.

CICS_EciSetProgramAliasExit

Function name: CICS_EciSetProgramAliasExit	Parameters: Anchor EciParms Program
--	---

Purpose

To allow the user to change the program name that the WorkLoad Manager of CICS Transaction Gateway for Windows uses for load balancing

This exit is only available when the WorkLoad Manager is enabled.

When called

Immediately before the WorkLoad Manager tries to select a server for an ECI program to connect to.

Parameters**Anchor**

Input parameter. The pointer set up by CICS_EciInitializeExit.

ECIParms

ECI parameter block.

Program

The alias name of the ECI program that the WorkLoad Manager will use for load balancing.

Return codes**CICS_EXIT_OK**

The ECI continues processing the request.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The ECI writes a CICS Transaction Gateway trace record, and then continues processing the request.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The ECI writes a CICS Transaction Gateway trace record, and then continues processing the request.

user-defined

User-defined return codes must have a value not less than CICS_EXIT_USER_BASE. The ECI writes a CICS Transaction Gateway trace record, and then continues processing the request.

C EPI exits reference

In this section the following exits are discussed:

- **CICS_EpiInitializeExit**
- **CICS_EpiTerminateExit**
- **CICS_EpiAddTerminalExit**
- **CICS_EpiTermIdExit**
- **CICS_EpiTermIdInfoExit**
- **CICS_EpiStartTranExtendedExit**
- **CICS_EpiStartTranExit**
- **CICS_EpiReplyExit**
- **CICS_EpiDelTerminalExit**
- **CICS_EpiGetEventExit**
- **CICS_EpiSystemIdExit**
- **CICS_EpiTranFailedExit**

Table 3 on page 194 summarizes the exit names, the parameters passed to each exit, and the possible return codes.

Table 3. Summary of EPI exits

Function name	Parameters	Return codes:
CICS_EpiInitializeExit	Version Anchor	CICS_EXIT_OK CICS_EXIT_NO_EXIT CICS_EXIT_CANT_INIT_EXITS user-defined
CICS_EpiTerminateExit	Anchor	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_STORAGE user-defined
CICS_EpiAddTerminalExit	Anchor NameSpace System NetName DevType	CICS_EXIT_OK CICS_EXIT_DONT_ADD_TERMINAL CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
CICS_EpiTermIdExit	Anchor TermIndex System	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
CICS_EpiTermIdInfoExit	Anchor Version TermIndex EpiDetails	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
CICS_EpiStartTranExtendedExit	Anchor TermIndex TransId Data Size	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
CICS_EpiStartTranExit	Anchor TransId Data Size	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
CICS_EpiReplyExit	Anchor TermIndex Data Size	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined

Table 3. Summary of EPI exits (continued)

Function name	Parameters	Return codes:
CICS_EpiDelTerminalExit	Anchor TermIndex	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined
CICS_EpiGetEventExit	Anchor TermIndex Wait Event	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined
CICS_EpiSystemIdExit	Anchor NameSpace System NetName DevType FailedSystem Reason SubReason UserId PassWord	CICS_EXIT_OK CICS_EXIT_DONT_ADD_TERMINAL CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined
CICS_EpiTranFailedExit	Anchor TermIndex Wait Event	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined

CICS_EpiInitializeExit

Function name: CICS_EpiInitializeExit	Parameters: Version Anchor
--	----------------------------------

Purpose

To allow the user to set up an exit environment.

When called

On each invocation of **CICS_EpiInitialize**, after the EPI has validated the parameters.

Parameters

Version

Input parameter. The version of the EPI under which the exit is running.

Anchor

Output parameter. A pointer to a pointer that will be passed to the EPI exits. The second pointer is not used by the EPI; it is passed to the exits as supplied. You can acquire storage in this exit and pass its address to the other exits.

Return codes

CICS_EXIT_OK

The EPI continues processing this request, calling the exits where appropriate.

CICS_EXIT_NO_EXIT

The EPI continues processing this request, but does not call any more exits.

CICS_EXIT_CANT_INIT_EXITS

The EPI writes a CICS Transaction Gateway trace record, and then continues processing this request, but does not call any more exits.

user-defined

User-defined return codes must have a value not less than CICS_EXIT_USER_BASE. The EPI writes a CICS Transaction Gateway trace record, and then continues processing this request, but does not call any more exits.

CICS_EpiTerminateExit

Function name: CICS_EpiTerminateExit	Parameters: Anchor
--	------------------------------

Purpose

To allow the user to clean up the exit environment. Any storage acquired by CICS_EpiInitializeExit must be released in this exit.

When called

On each invocation of CICS_EpiTerminate, after the EPI has validated the parameters.

Parameters

Anchor

Input parameter. The pointer set up by `CICS_EpiInitializeExit`.

Return codes

CICS_EXIT_OK

Termination continues.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then continues with termination.

CICS_EXIT_BAD_STORAGE

CICS detected a storage error. The EPI writes a CICS Transaction Gateway trace record, and then continues with termination.

user-defined

User-defined return codes must have a value not less than `CICS_EXIT_USER_BASE`. The EPI writes a CICS Transaction Gateway trace record, and then continues with termination.

CICS_EpiAddTerminalExit

Function name:	Parameters:
CICS_EpiAddTerminalExit	Anchor
	NameSpace
	System
	NetName
	DevType

Purpose

To allow the user to select a server, or override the one passed to `CICS_EpiAddTerminal` or `CICS_EpiAddExTerminal` in the **System** parameter.

When called

On each invocation of `CICS_EpiAddTerminal` or `CICS_EpiAddExTerminal`, after the EPI has validated the parameters.

Parameters

Anchor

Input parameter. The pointer storage set up by `CICS_EpiInitializeExit`.

NameSpace

Input-output parameter. On input, its value depends on the value supplied for the **NameSpace** parameter of the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call to which this exit relates:

- If a null pointer was supplied, this input is a pointer to a null string.
- If a non-null pointer was supplied, the **Namespace** input parameter points to a copy of this data.

On output, it will be used by the EPI in the same way as the value specified on the call would have been used.

System

Input-output parameter. On input, it is the value supplied for the **System** parameter of the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call to which this exit relates. On output, it will be used by the EPI in the same way as the value specified on the call would have been used.

NetName

Input-output parameter. On input, it is the value supplied for the **NetName** parameter of the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call to which this exit relates. On output, it will be used by the EPI in the same way as the value specified on the call would have been used.

DevType

Input-output parameter. On input, it is the value supplied for the **DevType** parameter of the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call to which this exit relates. On output, it will be used by the EPI in the same way as the value specified on the call would have been used.

Return codes

CICS_EXIT_OK

Processing continues with the output values of **NameSpace**, **System**, **NetName**, and **DevType**.

CICS_EXIT_DONT_ADD_TERMINAL

The **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** is ended with a return code of **CICS_EPI_ERR_FAILED**.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then continues as for **CICS_EXIT_OK**.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The EPI writes a CICS Transaction Gateway trace record, and then continues as for CICS_EXIT_OK.

user-defined

User-defined return codes must have a value not less than CICS_EXIT_USER_BASE. The EPI writes a CICS Transaction Gateway trace record, and then continues as for CICS_EXIT_OK.

Notes

Note on selection of systems:

If the calling application does not specify system name in its parameter list, then it is expecting that the system will be dynamically selected, and the exit may safely select the system.

If however the calling application specifies a system name, then it may not be expecting the target system to change and application errors could result. In this case the exit would generally not specify a replacement system, with the result that the specified or default system name, device type, etc. is to be used. If the exit chooses to change the selected system in this situation, then it may do so, but the following should be borne in mind.

- The exit routine must be sensitive to whether or not the modification of the target system will cause errors in the EPI application running on the client.
- The exit routine must maintain a knowledge base, keyed on appropriate data available to it, to enable it to determine whether this modification is acceptable to the client application.

CICS_EpiAddTerminalExit and CICS_EpiSystemIdExit:

The relationship between these exits is as follows. The exits will get multiple chances to make a selection of the system. The first chance will always occur on the **CICS_EpiAddTerminalExit**. This exit will only receive the parameters passed by the application to **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal**. If an error occurs when CICS tries to add the terminal (whether or not the exit has made a selection) then **CICS_EpiSystemIdExit** will be called. **CICS_EpiSystemIdExit** will additionally be passed the error that occurred on the attempt to add the terminal, and will get a chance to correct the error. This continues to occur until either a terminal is successfully added, or until **CICS_EpiSystemIdExit** signals to give up.

If no error occurs on the attempt to add the terminal, then **CICS_EpiSystemIdExit** will not be called.

CICS_EpiTermIdExit

Function name:
CICS_EpiTermIdExit

Parameters:
Anchor
TermIndex
System

Purpose

To allow the user to know the terminal index allocated after a successful call to **CICS_EpiAddTerminal**.

CICS_EpiTermIdExit is provided for compatibility with older applications only. All new applications that use the EPI exits should use **CICS_EpiTermIdInfoExit** instead.

When called

On each invocation of **CICS_EpiAddTerminal**, after the server has allocated the terminal.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

TermIndex

Input parameter. This is the terminal index for the terminal resource just reserved or installed.

System

Input parameter. A pointer to a null-terminated string that specifies the name of the server in which the terminal resource has been reserved or installed.

Return codes

CICS_EXIT_OK

Processing continues.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then continues as for CICS_EXIT_OK.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The EPI writes a CICS Transaction Gateway trace record, and then continues as for CICS_EXIT_OK.

user-defined

User-defined return codes must have a value not less than CICS_EXIT_USER_BASE. The EPI writes a CICS Transaction Gateway trace record, and then continues as for CICS_EXIT_OK.

CICS_EpiTermIdInfoExit

Function name: CICS_EpiTermIdInfoExit	Parameters: Anchor Version TermIndex EpiDetails
---	--

Purpose

To allow the user to retrieve information about the current terminal.

When called

Immediately after a CICS terminal has been installed

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

Version

Input parameter. The EPI version.

TermIndex

Input parameter. The index of the terminal being installed.

EpiDetails

Input parameter. A pointer to the **CICS_EpiDetails_t** structure, containing details about the terminal being installed.

Return codes

CICS_EXIT_OK

Processing continues.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then continues as for CICS_EXIT_OK.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The EPI writes a CICS Transaction Gateway trace record, and then continues as for CICS_EXIT_OK.

user-defined

User-defined return codes must have a value not less than CICS_EXIT_USER_BASE. The EPI writes a CICS Transaction Gateway trace record, and then continues as for CICS_EXIT_OK.

CICS_EpiStartTranExtendedExit

Function name: CICS_EpiStartTranExtendedExit	Parameters: Anchor TermIndex TransId Data Size
--	--

Purpose

To allow the user to see when a transaction is started, for information gathering purposes. This exit does not select a system, and has no return data.

When called

On invocation of **CICS_EpiStartTran**, after the EPI has validated the parameters.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

TermIndex

Input parameter. The value supplied by the TermIndex parameter of the CICS_EpiReply call to which this exit relates.

TransId

Input parameter. The value supplied for the **TransId** parameter of the **CICS_EpiStartTran** call to which this exit relates.

Data Input parameter. The value supplied for the **Data** parameter of the **CICS_EpiStartTran** call to which this exit relates.

Size Input parameter. The value supplied for the **Size** parameter of the **CICS_EpiStartTran** call to which this exit relates.

Return codes

CICS_EXIT_OK

Processing of the **CICS_EpiStartTran** call continues.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiStartTran** call continues.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiStartTran** call continues.

user-defined

User-defined return codes must have a value not less than **CICS_EXIT_USER_BASE**. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiStartTran** call continues.

CICS_EpiStartTranExit

Function name: CICS_EpiStartTranExit	Parameters: Anchor TransId Data Size
--	---

Purpose

To allow the user to see when a transaction is started, for information gathering purposes. This exit will not select a system, and has no return data.

When called

On invocation of **CICS_EpiStartTran**, after the EPI has validated the parameters.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

TransId

Input parameter. The value supplied for the **TransId** parameter of the **CICS_EpiStartTran** call to which this exit relates.

Data Input parameter. The value supplied for the **Data** parameter of the **CICS_EpiStartTran** call to which this exit relates.

Size Input parameter. The value supplied for the **Size** parameter of the **CICS_EpiStartTran** call to which this exit relates.

Return codes

CICS_EXIT_OK

Processing of the **CICS_EpiStartTran** call continues.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiStartTran** call continues.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiStartTran** call continues.

user-defined

User-defined return codes must have a value not less than **CICS_EXIT_USER_BASE**. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiStartTran** call continues.

CICS_EpiReplyExit

Function name: CICS_EpiReplyExit	Parameters: Anchor TermIndex Data Size
--	---

Purpose

To allow the user to see when a transaction is replied to, for information gathering purposes.

When called

On invocation of **CICS_EpiReply**, after the EPI has validated the parameters.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

TermIndex

Input parameter. The value supplied for the **TermIndex** parameter of the **CICS_EpiReply** call to which this exit relates.

Data Input parameter. The value supplied for the **Data** parameter of the **CICS_EpiReply** call to which this exit relates.

Size Input parameter. The value supplied for the **Size** parameter of the **CICS_EpiReply** call to which this exit relates.

Return codes

CICS_EXIT_OK

Processing of the **CICS_EpiReply** call continues.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiReply** call continues.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiReply** call continues.

user-defined

User-defined return codes must have a value not less than **CICS_EXIT_USER_BASE**. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiReply** call continues.

CICS_EpiDelTerminalExit

Function Name: CICS_EpiDelTerminalExit	Parameters: Anchor TermIndex
--	---

Purpose

To allow the user to clean up any terminal-related data structures.

When called

On invocation of **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal**, after the EPI has validated the parameters. To allow the user to clean up any terminal-related data structures.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

TermIndex

Input parameter. The value supplied for the **TermIndex** parameter of the **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal** call to which this exit relates.

Return codes

CICS_EXIT_OK

Processing of the **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal** call continues.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal** call continues.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal** call continues.

user-defined

User-defined return codes must have a value not less than **CICS_EXIT_USER_BASE**. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal** call continues.

CICS_EpiGetEventExit

Function name:
CICS_EpiGetEventExit

Parameters:
Anchor
TermIndex
Wait
Event

Purpose

To allow the user to collect data relating to the event that has arrived.

When called

Immediately before **CICS_EpiGetEvent** returns to the caller. The exit can then examine the data returned, time the response from the system, etc.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

TermIndex

Input parameter. The value to be returned to the application in the **TermIndex** parameter of the **CICS_EpiGetEvent** call to which this exit relates.

Wait Input parameter. The value supplied for the **Wait** parameter of the **CICS_EpiGetEvent** call to which this exit relates.

Event Input parameter. The value to be returned to the application in the **Event** parameter of the **CICS_EpiGetEvent** call to which this exit relates.

Return codes

CICS_EXIT_OK

Processing of the **CICS_EpiGetEvent** call continues.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiGetEvent** call continues.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiGetEvent** call continues.

user-defined

User-defined return codes must have a value not less than **CICS_EXIT_USER_BASE**. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiGetEvent** call continues.

CICS_EpiSystemIdExit

Function name: CICS_EpiSystemIdExit	Parameters: Anchor NameSpace System NetName DevType FailedSystem Reason SubReason UserId PassWord
---	--

Purpose

To allow the user to supply a new system name when the value supplied for **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** was invalid.

When called

Immediately before **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** returns to the application when an error occurred while trying to add the terminal. The error can be **CICS_EPI_ERR_SYSTEM**, **CICS_EPI_ERR_FAILED**, or **CICS_EPI_ERR_SERVER_DOWN**. It occurs whether or not **CICS_EpiAddTerminalExit** or **CICS_EpiAddExTerminal** has been called previously.

Note: On some systems the completion of **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** is returned to the application asynchronously, and in this case this exit will be called asynchronously.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

NameSpace

Input-output parameter. The **NameSpace** parameter used in the failed **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal**.

System

Input-output parameter. The **System** parameter used in the failed **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal**.

NetName

Input-output parameter. The **NetName** parameter used in the failed **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal**.

DevType

Input-output parameter. The **DevType** parameter used in the failed **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal**.

FailedSystem

Input parameter. The identifier of the system on which the failure occurred.

Reason

Input parameter. The reason for the failure: **CICS_EPI_ERR_SYSTEM** or **CICS_EPI_ERR_FAILED**.

SubReason

Input parameter. More about the failure.

UserId

Output parameter. Not used.

PassWord

Output parameter. Not used.

Return codes**CICS_EXIT_OK**

The EPI will retry the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call using the values specified as output of this exit. Note that in this case the considerations described in “**CICS_EpiAddTerminalExit**” on page 197 apply.

CICS_EXIT_DONT_ADD_TERMINAL

The **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** is ended with a return code of **CICS_EPI_ERR_FAILED**.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then the error that caused the exit to be called is returned to the application.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The EPI writes a CICS Transaction Gateway trace record, and then the error that caused the exit to be called is returned to the application.

user-defined

User-defined return codes must have a value not less than

CICS_EXIT_USER_BASE. The EPI writes a CICS Transaction Gateway trace record, and then the error that caused the exit to be called is returned to the application.

CICS_EpiTranFailedExit

Function Name: CICS_EpiTranFailedExit	Parameters: Anchor TermIndex Wait Event
---	---

Purpose

To allow the user to collect data when a transaction abends or a terminal fails.

When called

Immediately before **CICS_EpiGetEvent** returns to the caller, with or without **GetEventExit**, when the event is CICS_EPI_EVENT_END_TRAN, and the **AbendCode** field is not blank.

Note that there are some failures on remote systems that can occur and will simply cause the presentation of a 3270 data stream with an error message and no abend code in the CICS_EPI_EVENT_END_TRAN. This error message may not even occur on the same event as the CICS_EPI_EVENT_END_TRAN. If the exit requires to handle this situation, it may monitor it through **CICS_EpiGetEventExit** and scan the appropriate 3270 data streams.

Parameters

Anchor

Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

TermIndex

Input parameter. The value to be returned to the application in the **TermIndex** parameter of the **CICS_EpiGetEvent** call to which this exit relates.

Wait Input parameter. The value supplied for the **Wait** parameter of the **CICS_EpiGetEvent** call to which this exit relates.

Event Input parameter. The value to be returned to the application in the **Event** parameter of the **CICS_EpiGetEvent** call to which this exit relates.

Return codes

CICS_EXIT_OK

Processing of the **CICS_EpiGetEvent** call continues.

CICS_EXIT_BAD_ANCHOR

CICS detected an invalid anchor field. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiGetEvent** call continues.

CICS_EXIT_BAD_PARM

CICS detected an invalid parameter. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiGetEvent** call continues.

user-defined

User-defined return codes must have a value not less than **CICS_EXIT_USER_BASE**. The EPI writes a CICS Transaction Gateway trace record, and then processing of the **CICS_EpiGetEvent** call continues.

Chapter 6. Statistical API reference

The information center contains statistical API reference material supplied with CICS Transaction Gateway.

| The documentation is also included as part of Installing CICS Transaction
| Gateway in <install_path>/docs/ctgstatsdoc.zip. To view the documentation,
| expand the contents of the compressed file into a suitable directory and view
| the index.htmlfile with a Web browser.

The material is not included with the CICS Universal Client.

Appendix A. COM Global Constants

Constants are provided in the type libraries for the Client daemon COM libraries. The libraries are in CCLIECI.DLL and CCLIEPI.DLL.

If you are using Visual Basic, you can look at the definitions in the type libraries by using Visual Basic Object viewer or another type library viewer.

If you are using VBScript, you cannot access the enumerations defined in the type library; use the numeric values provided here.

The exception code constants are listed in Appendix D, "COM Error Code References," on page 225.

Appendix B. COM EPI Specific Constants

Synchronization Types

Table 4. Synchronization types

VB Enumeration	Value	Description
cclSync	0	Synchronous call type
cclDsync	1	Deferred synchronous call type

CclEPI States

Table 5. CclEPI States

VB Enumeration	Value	Description
cclEPIActive	0	EPI initialized OK
cclDiscon	1	EPI Terminated
cclEPIError	2	EPI failed to initialize, handle exception for more information

CclSession States

Table 6. CclSession States

VB Enumeration	Value	Description
cclSessionIdle	0	Idle, client needs to initiate transaction
cclSessionServer	1	Waiting for server
ccISessionClient	2	Waiting for Client daemon to respond
cclSessionDiscon	3	Disconnected
cclSessionError	4	Session Error, handle exception for more information

CclTerminal States

Table 7. CclTerminal States

VB Enumeration	Value	Description
cclInit	0	Terminal defined but not installed

Table 7. CclTerminal States (continued)

VB Enumeration	Value	Description
cclActive	1	Terminal connected (not used)
cclIdle	2	Idle, Client daemon needs to initiate transaction
cclServer	3	Waiting for server
cclClient	4	Waiting for client to respond
cclDiscon	5	Disconnected
cclError	6	Terminal error, handle exception for more information

CclTerminal ATI States

Table 8. CclTerminal ATI states

VB Enumeration	Value	Description
cclATIEnabled	0	ATIs are allowed
cclATIDisabled	1	ATIs are not allowed

CclTerminal EndTermReasons

Table 9. CclTerminal ATI states

VB Enumeration	Value	Description
cclSignoff	0	Disconnect request or user has signed off the terminal
cclShutdown	1	The CICS server has been shut down
cclOutOfService	2	The terminal has been switched to out of use
cclUnknown	3	An unknown situation as occurred
cclFailed	4	The terminal failed to disconnect
cclNotDiscon	5	The terminal is not disconnected

CclTerminal Sign-on Types

Table 10. CclTerminal Sign-on Types

VB Enumeration	Value	Description
cclSignonCapable	0	Terminal supports sign-on transaction

Table 10. CclTerminal Sign-on Types (continued)

VB Enumeration	Value	Description
cclSignonIncapable	1	Terminal does not support sign-on transaction
cclSignonUnknown	2	Terminal sign-on capability is unknown

CclScreen AID key codes

Table 11. CclScreen AID key codes

VB Enumeration	Value	Description
cclEnter	0	Enter key
cclClear	1	Clear key
cclPA1	2	Program Attention key 1
cclPA2	3	Program Attention key 2
cclPA3	4	Program Attention key 3
cclPF1	5	Program Function key 1
cclPF2	6	Program Function key 2
cclPF3	7	Program Function key 3
cclPF4	8	Program Function key 4
cclPF5	9	Program Function key 5
cclPF6	10	Program Function key 6
cclPF7	11	Program Function key 7
cclPF8	12	Program Function key 8
cclPF9	13	Program Function key 9
cclPF10	14	Program Function key 10
cclPF11	15	Program Function key 11
cclPF12	16	Program Function key 12
cclPF13	17	Program Function key 13
cclPF14	18	Program Function key 14
cclPF15	19	Program Function key 15
cclPF16	20	Program Function key 16
cclPF17	21	Program Function key 17
cclPF18	22	Program Function key 18
cclPF19	23	Program Function key 19
cclPF20	24	Program Function key 20

Table 11. CclScreen AID key codes (continued)

VB Enumeration	Value	Description
cclPF21	25	Program Function key 21
cclPF22	26	Program Function key 22
cclPF23	27	Program Function key 23
cclPF24	28	Program Function key 24

CclField Protected State Attributes

Table 12. CclField Protected state attributes

VB Enumeration	Value	Description
cclProtect	0	Protected Field (cannot be modified)
cclUnprotect	1	Unprotected (input) field

CclField Numeric Attributes

Table 13. CclField Numeric Attributes

VB Enumeration	Value	Description
cclAlphanumeric	0	Alphanumeric input field
cclNnumeric	1	Numeric input field

CclField Intensity Attributes

Table 14. CclField Intensity attributes

VB Enumeration	Value	Description
cclNormal	0	Normal display
cclIntense	1	Intensified display
cclDark	2	Non-display field

CclField Modified Attributes

Table 15. CclField Modified Attributes

VB Enumeration	Value	Description
cclUnmodified	0	Field has not been changed
cclModified	1	Field has been changed

CclField Highlight Attributes

Table 16. CclField Highlight attributes

VB Enumeration	Value	Description
cclHltDefault	0	Default field text highlighting
cclHltNormal	1	Field text highlight as specified by 3270 base attribute
cclHltBlink	2	Blinking text
cclHltReverse	3	Reverse video text
cclHltUnderscore	4	Underscored text
cclHltIntense	5	High intensity text

CclField Transparency Attributes

Table 17. CclField Transparency attributes

VB Enumeration	Value	Description
cclTrnDefault	0	Default (opaque) field background
cclTrnOr	1	Transparent field background (OR)
cclTrnXor	2	Transparent field background (XOR)
cclTrnOpaque	3	Opaque field background

CclField Color Attributes

Table 18. CclField Color attributes

VB Enumeration	Value	Description
cclDefaultColor	0	
cclBlue	1	
cclRed	2	
cclPink	3	
cclGreen	4	
cclCyan	5	
cclYellow	6	
cclNeutral	7	
cclBlack	8	

Table 18. CclField Color attributes (continued)

VB Enumeration	Value	Description
cclDarkBlue	9	
cclOrange	10	
cclPurple	11	
cclPaleGreen	12	
cclPaleCyan	13	
cclGray	14	
cclWhite	15	

Appendix C. COM ECI Constants

Synchronization Types

Table 19. Synchronization types

VB Enumeration	Value	Description
cclSync	0	Synchronous call type
cclDsync	1	Deferred synchronous call type

Flow status types

Table 20. Flow status types

VB Enumeration	Value	Description
cclInactive	0	Flow is inactive
cclLink	1	Flow is currently making a link call
cclBackout	2	Flow is currently backing out a UOW
cclCommit	3	Flow is currently committing a UOW
cclStatus	4	Flow is requesting status
cclChanged	5	Flow is requesting a status change
cclCancel	6	Flow is requesting a status cancel

Connection Status Codes

Table 21. Connection status code

VB Enumeration	Value	Description
cclUnknown	0	The CICS server status is unknown
cclAvailable	1	The CICS server status is available
cclUnavailable	2	The CICS server status is unavailable

Appendix D. COM Error Code References

Enumeration	Value	Description	ECI	EPI
cclNoError	0	No error occurred	Yes	Yes
cclBufferOverflow	1	Attempted to increase a CclBuf object which isn't Extensible	Yes	
cclMultipleInstance	2	Attempted to create more than one ECI object	Yes	
cclActiveFlow	3	Current Flow is still active, you cannot use this flow until it is inactive	Yes	
cclActiveUOW	4	Current UOW is still active, you need to backout or commit.	Yes	
cclSyncType	5	Incorrect synchronization type for method call.	Yes	Yes
cclDataLength	9	CommArea > 32768 Bytes or inbound 3270 data stream too large for Terminal Buffer size.	Yes	Yes
cclNoCICS	10	The Client daemon is unavailable, or the server implementation is unavailable, or a logical unit of work was to be begun, but the CICS server specified is not available. No resources have been updated	Yes	Yes
cclCICSDied	11	A logical unit of work was to be begun or continued, but the CICS server was no longer available. If this is a link call with an active UOW, the changes are backed out. If This was a UOW Commit or the application cannot determine whether the changes have been committed or backed out, and must log this condition to aid future manual recovery	Yes	
cclNoReply	12	There was no outstanding reply	Yes	
cclTransaction	13	ECI Program Abended	Yes	
cclSystemError	14	Unknown internal error occurred	Yes	Yes
cclResource	15	The server implementation or the Client daemon did not have enough resources to complete the request e.g. insufficient SNA sessions.	Yes	Yes

Enumeration	Value	Description	ECI	EPI
cclMaxUOWs	16	A new logical unit of work was being created, but the application already has as many outstanding logical units of work as the configuration will support.	Yes	
cclUnknownServer	17	The requested server could not be located	Yes	Yes
cclSecurity	18	You did not supply a valid combination of user ID and password, though the server expects it.	Yes	Yes
cclMaxServers	19	You attempted to start requests to more servers than your configuration allows. You should consult the documentation for your Client daemon or server to see how to control the number of servers you can use.	Yes	Yes
cclMaxRequests	20	There were not enough communication resources to satisfy the request. You should consult the documentation for your Client daemon or server to see how to control communication resources	Yes	Yes
cclRolledBack	21	An attempt was made to commit a logical unit of work, but the server was unable to commit the changes, and backed them out instead	Yes	
cclParameter	22	Incorrect parameter supplied	Yes	Yes
cclInvalidState	23	The Object is not in the correct state to invoke the method, e.g. terminal object still in server state and an attempt to send data is made.	Yes	Yes
ccltransId	24	Null transid supplied or returned for a pseudo conversational transaction		Yes
cclInitEPI	25	No EPI object or EPI failed to initialize correctly		Yes
cclConnect	26	Unexpected error trying to add the terminal		Yes
ccldata stream	27	Unsupported Data Stream		Yes
cclInvalidMap	28	Map definition and Screen do not match		Yes
cclClass	29	Unknown internal Class error occurred.	Yes	Yes
cclStartTranFailure	30	Transaction failed to start		Yes
cclTimeout	31	Timeout occurred before response from Server	Yes	Yes
cclNoPassword	32	The object's password is null.	Yes	Yes

Enumeration	Value	Description	ECI	EPI
cclNoUserid	33	The object's userid is null	Yes	Yes
cclNullNewPassword	34	The provided password is null	Yes	Yes
cclPemNotSupported	35	The CICS Server does not support the Password Expiry Management facilities. The method cannot be used	Yes	Yes
cclPemNotActive	36	Password Expiry Management is not active	Yes	Yes
cclPasswordExpired	37	The password has expired. No information has been returned	Yes	Yes
cclPasswordInvalid	38	The password is invalid.	Yes	Yes
cclPasswordRejected	39	Change password failed because the password doesn't conform to standards defined	Yes	Yes
cclUseridInvalid	40	The userid is unknown	Yes	Yes
cclInvalidTermid	41	Invalid Terminal ID		Yes
cclInvalidModelId	42	Invalid Model/Type		Yes
cclnot3270	43	Not a 3270 device		Yes
cclinvalidCCSid	44	Invalid CCSid		Yes
cclServerBusy	45	CICS server is busy		Yes
cclSignonNotPoss	46	The server does not allow the terminal to be installed as sign-on capable.		Yes

Appendix E. Java encodings

This appendix lists the supported Java encodings. A canonical name is converted to the corresponding CCSid so that a CICS server can determine in which code page a data stream can be located.

Note:

1. Your CICS server must support EPI Version 2 for the encodings to be implemented.
2. Check your CICS Server documentation to find out which CCSids your server supports.

Canonical name	Description	CCSid
Cp1252	Windows Latin-1	5348
ISO8859_1	ISO 8859-1, Latin alphabet No. 1	819
UTF8	Eight-bit Unicode Transformation format	1208
ASCII	American Standard Code for Information Interchange	437
Big5	Big 5, Traditional Chinese	950
Cp037	USA, Canada (Bilingual, French), Netherlands, Portugal, Brazil, Australia	37
Cp273	IBM Austria, Germany	273
Cp277	IBM Denmark, Norway	277
Cp278	IBM Finland, Sweden	278
Cp280	IBM Italy	280
Cp284	IBM Catalan/Spain, Spanish Latin America	284
Cp285	IBM United Kingdom, Ireland	285
Cp297	IBM France	297
Cp420	IBM Arabic	420
Cp424	IBM Hebrew	424
Cp437	MS-DOS United States, Australia, New Zealand, South Africa	437
Cp500	EBCDIC 500V1	500
Cp838	IBM Thailand extended SBCS	9030
Cp850	MS-DOS Latin-1	850
Cp852	MS-DOS Latin-2	852
Cp855	IBM Cyrillic	855

Canonical name	Description	CCSid
Cp856	IBM Hebrew	856
Cp857	IBM Turkish	857
Cp858	Variant of Cp850 with euro character	858
Cp862	PC Hebrew	862
Cp864	PC Arabic	864
Cp865	MS-DOS Nordic	865
Cp866	MS-DOS Russian	866
Cp868	MS-DOS Pakistan	868
Cp869	IBM Modern Greek	869
Cp870	IBM Multilingual Latin-2	870
Cp871	IBM Iceland	871
Cp874	IBM Thai	9066
Cp875	IBM Greek	875
Cp918	IBM Pakistan (Urdu)	918
Cp921	IBM Latvia, Lithuania (AIX [®] , DOS)	921
Cp922	IBM Estonia (AIX, DOS)	922
Cp923	IBM Latin-9	923
Cp930	Japanese Katakana-Kanji mixed with 4370 UDC, superset of 5026	930
Cp933	Korean Mixed with 1880 UDC, superset of 5029	933
Cp935	Simplified Chinese Host mixed with 1880 UDC, superset of 5031	935
Cp937	Traditional Chinese Host mixed with 6204 UDC, superset of 5033	937
Cp939	Japanese Latin Kanji mixed with 4370 UDC, superset of 5035	939
Cp942	IBM OS/2 [®] Japanese, superset of Cp932	942
Cp942C	Variant of Cp942	942
Cp943	IBM OS/2 Japanese, superset of Cp932 and Shift-JIS	943
Cp943C	Variant of Cp943	943
Cp948	OS/2 Chinese (Taiwan) superset of 938	948
Cp949	PC Korean	949
Cp949C	Variant of Cp949	949
Cp950	PC Chinese (Hong Kong, Taiwan)	950
Cp964	AIX Chinese (Taiwan)	964
Cp970	AIX Korean	970

Canonical name	Description	CCSID
Cp1006	IBM AIX Pakistan (Urdu)	1006
Cp1025	IBM Multilingual Cyrillic: Bulgaria, Bosnia, Herzegovina, Macedonia (FYR)	1025
Cp1026	IBM Latin-5, Turkey	1026
Cp1097	IBM Iran (Farsi)/Persian	1097
Cp1098	IBM Iran (Farsi)/Persian	1098
Cp1112	IBM Latvia, Lithuania	1112
Cp1122	IBM Estonia	1122
Cp1123	IBM Ukraine	1123
Cp1124	IBM AIX Ukraine	1124
Cp1140	Variant of Cp037 with euro character	1140
Cp1141	Variant of Cp273 with euro character	1141
Cp1142	Variant of Cp277 with euro character	1142
Cp1143	Variant of Cp278 with euro character	1143
Cp1144	Variant of Cp280 with euro character	1144
Cp1145	Variant of Cp284 with euro character	1145
Cp1146	Variant of Cp285 with euro character	1146
Cp1147	Variant of Cp297 with euro character	1147
Cp1148	Variant of Cp500 with euro character	1148
Cp1149	Variant of Cp871 with euro character	1149
Cp1250	Windows Eastern European	5346
Cp1251	Windows Cyrillic	5347
Cp1253	Windows Greek	5349
Cp1254	Windows Turkish	5350
Cp1255	Windows Hebrew	5351
Cp1256	Windows Arabic	5352
Cp1257	Windows Baltic	5353
Cp1258	Windows Vietnamese	5354
Cp1381	IBM OS/2, DOS People's Republic of China (PRC)	1381
Cp1383	IBM AIX, People's Republic of China (PRC)	1383
EUC_CN	GB2312, EUC encoding, Simplified Chinese	1383
EUC_JP	JIS X 0201, 0208, 0212, EUC encoding, Japanese	954
EUC_KR	KS C 5601, EUC encoding, Korean	970

Canonical name	Description	CCSid
GBK	GBK, Simplified Chinese	1386
ISO8859_2	ISO 8859-2, Latin alphabet No. 2	912
ISO8859_5	ISO 8859-5, Latin/Cyrillic alphabet	915
ISO8859_6	ISO 8859-6, Latin/Arabic alphabet	1089
ISO8859_7	ISO 8859-7, Latin/Greek alphabet	813
ISO8859_8	ISO 8859-8, Latin/Hebrew alphabet	916
ISO8859_9	ISO 8859-9, Latin alphabet No. 5	920
ISO8859_15_FDIS	ISO 8859-15, Latin alphabet No. 9	923
JIS0201	JIS X 0201, Japanese	5050
JIS0208	JIS X 0208, Japanese	5050
JIS0212	JIS X 0212, Japanese	5050
EUC_TW	CNS 11643 (Plane 1-3), EUC encoding, Traditional Chinese	964
MS932	Windows Japanese	943
MS936	Windows Simplified Chinese	1386
MS949	Windows Korean	1363

Appendix F. C++ Exception Objects

All exception objects provide the following information

- Class Name
- Method Name
- Exception Code
- Exception Text
- Abend Code (ECI Only)
- Origin Point

The Class name can contain a trailing 'I', which implies it is an internally-contained class for the well known class. For example, CclFlowI is contained by CclFlow. If an internal class is reported the method reported might be an internal method, not an external one.

The Origin Point is a unique value which defines the exact point within the class library where the exception was generated. These are mainly useful for service.

The more important items of information are the Exception Code, Exception Text and Abend Code (ECI only). The following is a Summary of these Exception Codes and Text and whether they are relevant to ECI or EPI or both.

Table 22. Exception codes

Enumeration	Text	Description	ECI	EPI
Ccl::noError	no error	No error occurred	Yes	Yes
Ccl::bufferOverflow	buffer overflow	Attempted to increase a CclBuf object which isn't Extensible	Yes	
Ccl::multipleInstance	multiple instance	Attempted to create more than one ECI object	Yes	
Ccl::activeFlow	flow is active	Current Flow is still active, you cannot use this flow until it is inactive	Yes	
Ccl::activeUOW	UOW is active	Current UOW is still active, you need to backout or commit.	Yes	
Ccl::syncType	sync error	Incorrect synchronization type for method call.	Yes	Yes

Table 22. Exception codes (continued)

Enumeration	Text	Description	ECI	EPI
Ccl::threadCreate	thread create error	Internal thread creation error	Yes	Yes
Ccl::threadWait	thread wait error	Internal thread wait error	Yes	
Ccl::threadKill	thread kill error	Internal thread kill error	Yes	
Ccl::dataLength	data length invalid	CommArea > 32768 Bytes or inbound 3270 data stream too large for Terminal Buffer size.	Yes	Yes
Ccl::noCICS	no CICS	The Gateway is unavailable, or the server implementation is unavailable, or a logical unit of work was to be begun, but the CICS server specified is not available. No resources have been updated	Yes	Yes
Ccl::CICSDied	CICS died	A logical unit of work was to be begun or continued, but the CICS server was no longer available. If this is a link call with an active UOW, the changes are backed out. If this was a UOW Commit or Backout, the application cannot determine whether the changes have been committed or backed out, and must log this condition to aid future manual recovery.	Yes	
Ccl::noReply	no reply	There was no outstanding reply	Yes	
Ccl::transaction	transaction abend	ECI Program Abended	Yes	
Ccl::systemError	system error	Unknown internal error occurred	Yes	Yes
Ccl::resource	resource shortage	The server implementation or the Gateway did not have enough resources to complete the request e.g. insufficient SNA sessions.	Yes	Yes

Table 22. Exception codes (continued)

Enumeration	Text	Description	ECI	EPI
Ccl::maxUOWs	exceeded max UOWs	A new logical unit of work was being created, but the application already has as many outstanding logical units of work as the configuration will support.	Yes	
Ccl::unknownServer	unknown server	The requested server could not be located	Yes	Yes
Ccl::security	security error	You did not supply a valid combination of user ID and password, though the server expects it.	Yes	Yes
Ccl::maxServers	exceeded max servers	You attempted to start requests to more servers than your configuration allows. You should consult the documentation for your Gateway or server to see how to control the number of servers you can use.	Yes	Yes
Ccl::maxRequests	exceeded max requests	There were not enough communication resources to satisfy the request. You should consult the documentation for your Gateway or server to see how to control communication resources	Yes	Yes
Ccl::rolledBack	rolled back	An attempt was made to commit a logical unit of work, but the server was unable to commit the changes, and backed them out instead	Yes	
Ccl::parameter	parameter error	Incorrect parameter supplied	Yes	Yes
Ccl::invalidState	invalid object state	The Object is not in the correct state to invoke the method, e.g. terminal object still in server state and an attempt to send data is made.	Yes	Yes

Table 22. Exception codes (continued)

Enumeration	Text	Description	ECI	EPI
Ccl::transId	invalid transaction	Null transid supplied or returned for a pseudo conversational transaction		Yes
Ccl::initEPI	EPI not initialized	EPI has failed to initialize correctly or EPI object is missing		Yes
Ccl::connect	connection failed	Unexpected error trying to add the terminal		Yes
Ccl::data stream	3270 data stream error	Unsupported Data Stream		Yes
Ccl::invalidMap	map/screen mismatch	Map definition and Screen do not match		Yes
Ccl::cclClass	CICS class error	Unknown internal Class error occurred.	Yes	Yes
Ccl::startTranFailure	Start Transaction Failure	Transaction failed to start		Yes
Ccl::timeout	Timeout Occurred	Timeout occurred before response from Server	Yes	Yes
Ccl::noPassword	Password is Null	The object's password is null.	Yes	Yes
Ccl::noUserid	Userid is Null	The object's userid is null	Yes	Yes
Ccl::nullNewPassword	A NULL new password was supplied	The provided password is null	Yes	Yes
Ccl::pemNotSupported	PEM is not supported on the server	The CICS Server does not support the Password Expiry Management facilities. The method cannot be used	Yes	Yes
Ccl::pemNotActive	PEM is not active on the server	Password Expiry Management is not active	Yes	Yes
Ccl::passwordExpired	Password has expired	The password has expired. No information has been returned	Yes	Yes
Ccl::passwordInvalid	Password is invalid	The password is invalid.	Yes	Yes
Ccl::passwordRejected	New password was rejected	Change password failed because the password does not conform to standards defined	Yes	Yes
Ccl::useridInvalid	Userid unknown at server	The userid is unknown	Yes	Yes
Ccl::invalidTermid	Termid is invalid	The terminal ID is invalid		Yes

Table 22. Exception codes (continued)

Enumeration	Text	Description	ECI	EPI
Ccl:invalidModelid	Modelid is invalid	Invalid Model/Device Type		Yes
Ccl:not3270	Not a 3270 device	Not a 3270 device		Yes
Ccl:invalidCCSid	Code page (CCSid value) is invalid	Invalid CCSid		Yes
Ccl:serverBusy	Server is too busy	CICS server is busy		Yes
Ccl:signonNotPossible	Sign-on Capable terminal is not possible	The server does not allow the terminal to be installed as sign-on capable.		Yes

The product library and related literature

This information lists books on the CICS Transaction Gateway, and related topics.

CICS Transaction Gateway books

- *CICS Transaction Gateway: Windows Administration, SC34-6960-00*
This book describes the administration of the CICS Transaction Gateway for Windows.
- *CICS Transaction Gateway: UNIX and Linux Administration, SC34-6959-00*
This book describes the administration of the CICS Transaction Gateway for UNIX and Linux.
- *CICS Universal Client: Windows Administration, SC34-6963-00*
This book describes the administration of the CICS Transaction Gateway for Windows.
- *CICS Universal Client: UNIX and Linux Administration, SC34-6962-00*
This book describes the administration of the CICS Transaction Gateway for the Linux operating system.
- *CICS Transaction Gateway: z/OS Administration, SC34-6961-00*
This book describes the administration of the CICS Transaction Gateway for z/OS.
- *CICS Transaction Gateway: Messages, SC34-6964-00*
This online book lists and explains the error messages that can be generated by the CICS Transaction Gateway.
- *CICS Transaction Gateway: Programming Reference, SC34-6966-00*
This book provides information on the APIs of the programming languages supported by the CICS Transaction Gateway.
Additional HTML pages contain JAVA programming reference information.
- *CICS Transaction Gateway: Programming Guide, SC34-6965-00*
This introduction to programming for the CICS Transaction Gateway provides the information that you need to allow user applications to use CICS facilities in a client/server environment.

Sample configuration documents

Several sample configuration documents are available in portable document format (PDF). These documents give step-by-step guidance for configuring CICS Transaction Gateway for communication with CICS servers, using various protocols. They provide detailed instructions that extend the information in the CICS Transaction Gateway library.

Visit the following Web site:

www.ibm.com/software/cics/ctg

and follow the **Library** link.

Redbooks

The following International Technical Support Organization (ITSO) Redbook publication contains many examples of client/server configurations:

- *CICS Transaction Gateway V5 - The WebSphere® Connector for CICS*, SG24-6133
- *Revealed! Architecting Web Access to CICS*, SG24-5466
- *Enterprise JavaBeans for z/OS and OS/390® CICS Transaction Server V2.2*, SG24-6284
- *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401. This book provides information on developing J2EE applications.
- *Systems Programmer's Guide to Resource Recovery Services (RRS)*, SG24-6980-00. This book provides information on using RRS in various scenarios.
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide*, SG24-6517-00. This book provides information on using Communications Server for z/OS V1R2, including load balancing.
- *Redpaper: Transactions in J2EE*, REDP-3659-00. This redpaper provides a discussion of transactions in the J2EE environment, including one- and XA transactions.

You can obtain ITSO Redbooks® from a number of sources. For the latest information, see:

www.ibm.com/redbooks/

Other Useful Books

CICS Transaction Server publications

CICS Transaction Server for z/OS RACF Security Guide, SC34-6249

CICS interproduct communication

The following books describe the intercommunication facilities of the CICS server products:

- *CICS Family: Interproduct Communication*, SC34-6267
- *CICS Transaction Server for Windows V5.0 Intercommunication*, SC34-6209
- *CICS Transaction Server for z/OS CICS External Interfaces Guide*, SC34-6449
- *CICS Transaction Server for z/OS: Intercommunication Guide*, SC34-6448
- *CICS/VSE 2.3: Intercommunication Guide*, SC33-0701
- *CICS Transaction Server for iSeries® V5R2: Intercommunication*, SC41-5456
- *TXSeries® 5.1: CICS Intercommunication Guide*, SC09-4462

The first book above is a CICS family book containing a platform-independent overview of CICS interproduct communication.

CICS problem determination books

The following books describe the problem determination facilities of the CICS server products:

- *Transaction Server for Windows V5.0: Problem Determination*, GC34-6210
- *CICS Transaction Server for z/OS V3.1 CICS Problem Determination Guide*, SC34-6441
- *CICS/VSE 2.3 Problem Determination Guide*, SC33-0716
- *CICS Transaction Server for iSeries V5R2: Problem Determination*, SC41-5453
- *TXSeries V5.1: CICS Problem Determination Guide*, SC09-4465

You can find information on CICS products at the following Web site:

www.ibm.com/software/cics/ctg

Microsoft Windows publications

See this Web site:

www.microsoft.com/windows

APPC-related publications

IBM products

IBM Communications Server

See this Web page:

www.ibm.com/software/network/commsserver/library

IBM Personal Communications

See this Web page:

www.ibm.com/software/network/pcomm/library

Microsoft products

See this page Web:

<http://www.microsoft.com/hiserver/techinfo/productdoc/default.msp>

Systems Network Architecture (SNA)

- *SNA Formats*, GA27-3136
- *Systems Network Architecture Technical Overview*, GC30-3073
- *Guide to SNA over TCP/IP*, SC31-6527

Obtaining books from IBM

For information on books you can download, visit our Web site at:

www.ibm.com/software/cics/ctg

and follow the **Library** link.

Accessibility features for CICS Transaction Gateway

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully. The CICS Transaction Gateway supports keyboard-only operation. Topics on the following pages give details of accessibility features.

Visit the IBM Accessibility Center for more information about IBM's commitment to accessibility.

Documentation

See the Eclipse information center for an HTML version of the documentation.

Starting the Gateway daemon

You can start the Gateway daemon from a command prompt using a screen reader.

In some Telnet sessions, the screen reader might reread CICS Transaction Gateway log output or the command prompt after the CICS Transaction Gateway has started. This behavior is expected, and does not mean that the CICS Transaction Gateway has failed to start.

To determine if the CICS Transaction Gateway started correctly, check for the message:

```
'CTG6512I CICS Transaction Gateway initialization complete'.
```

If the CICS Transaction Gateway did not start successfully, this message is produced:

```
'CTG6513E CICS Transaction Gateway failed to initialize'.
```

Setting EPITerminal properties programmatically

The EPITerminal terminal properties sheet is not accessible. To set properties programmatically, use the `getTerminal()` method of the EPITerminal object and cast it to a Terminal object. For example, if `epiTerm` is an EPITerminal object, code something like the following:

```
Terminal term = (Terminal)epiTerm.getTerminal();
```

You can then use methods on the Terminal object to set these properties. To set the name for a CICS server named `YOURSERV`, code the following:

```
term.setServerName("YOURSERV");
```

See the Javadoc supplied with the product for full details of these setter methods.

cicsterm

Although `cicsterm` is accessible, it relies on the application that is being processed to define an accessible 3270 screen.

The bottom row of `cicsterm` contains status information. The following list shows this information, as it appears from left to right:

Status For example, **1B** is displayed while `cicsterm` is connecting to a server. Displayed at columns 1 – 3.

Terminal name

Also referred to as *LU Name*. Columns 4 – 7.

Action

For example, **X-System**, indicating that you cannot enter text in the terminal window because `cicsterm` is waiting for a response from the server. Columns 9 – 16.

Error number

Errors in the form CCLNNNN, relating to the CICS Transaction Gateway. Columns 17 – 24.

Server name

The server to which `cicsterm` is connected. Columns 27 – 35.

Upper case

An up arrow is displayed when the Shift key is pressed. Column 42.

Caps Lock

A capital A is displayed when Caps Lock is on. Column 43.

Insert on

The caret symbol (^) is displayed if text will be inserted, rather than overwriting existing text. If you have difficulty seeing the caret, change the font face and size, or use a screen magnifier to increase the size of the status line. Column 52.

Cursor position

The cursor position, in the form ROW/COLUMN, where ROW is a two-digit number, and COLUMN a three-digit number. The top left of the screen is 01/001. Column 75–80.

Note: You might need to change the default behavior of your screen reader if it reads only the last digit of the cursor position. Customize your screen reader to specify that columns 75–80 of

the status row are to be treated as one field. This will cause the full area to be read when any digit changes.

The cicsterm -? command

After issuing the cicsterm -? command, use the up arrow key to move from the **OK** button to the list of messages. Use the up and down arrow keys to move through the messages. Press **Tab** and then **Enter** when done.

Glossary

This glossary defines special terms used in the CICS Transaction Gateway library.

3270 emulation

The use of software that enables a client to emulate an IBM 3270 display station or printer, and to use the functions of an IBM host system.

abnormal end of task (abend)

The termination of a task, job, or subsystem because of an error condition that recovery facilities cannot resolve.

Advanced program-to-program communication (APPC)

An implementation of the SNA/SDLC LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs. The Client daemon uses APPC to communicate with CICS server systems.

APAR See *Authorized program analysis report*.

API Application programming interface.

applet A small application program that performs a specific task and is usually portable between operating systems. Often written in Java, applets can be downloaded from the Internet and run in a Web browser.

application identifier

The name by which a CICS system is known in a network of interconnected CICS systems. CICS Transaction Gateway application identifiers do not need to be defined in SYS1.VTAMLST. The CICS APPLID is specified in the APPLID system initialization parameter.

application programming interface (API)

A functional interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

APPLID

See *application identifier*.

ARM See *automatic restart management*.

Authorized program analysis report (APAR)

A request for correction of a defect in a current release of an IBM-supplied program.

ATI See *automatic transaction initiation*.

attach In SNA, the request unit that flows on a session to initiate a conversation.

Attach Manager

The component of APPC that matches attaches received from remote computers to accepts issued by local programs.

autoinstall

A method of creating and installing resources dynamically as terminals log on, and deleting them at logoff.

automatic restart manager

A z/OS recovery function that can improve the availability of specific batch jobs or started tasks, and therefore result in faster resumption of productive work. Acronym: ARM.

automatic transaction initiation (ATI)

The initiation of a CICS transaction by an internally generated request, for example, the issue of an EXEC CICS START command or the reaching of a transient data trigger level. CICS resource definition can associate a trigger level and a transaction with a transient data destination. When the number of records written to the destination reaches the trigger level, the specified transaction is automatically initiated.

bean A definition or instance of a JavaBeans™ component. See also *JavaBeans*.

bean-managed transaction

A transaction where the J2EE bean itself is responsible for administering transaction tasks such as committal or rollback. See also *container-managed transaction*.

BIND command

In SNA, a request to activate a session between two logical units (LUs).

business logic

The part of a distributed application that is concerned with the application logic rather than the user interface of the application. Compare with *presentation logic*.

CA See *certificate authority*.

callback

A way for one thread to notify another application thread that an event has happened.

certificate authority

In computer security, an organization that issues certificates. The

certificate authority authenticates the certificate owner's identity and the services that the owner is authorized to use. It issues new certificates and revokes certificates from users who are no longer authorized to use them.

change-number-of-sessions (CNOS)

An internal transaction program that regulates the number of parallel sessions between the partner LUs with specific characteristics.

channel

A channel is a set of containers, grouped together to pass data to CICS. There is no limit to the number of containers that can be added to a channel, and the size of individual containers is limited only by the amount of storage that you have available.

CICS connectivity components

A generic reference to the Client daemon, EXCI, and the IPIC protocol.

CICS on System/390®

A generic reference to the products CICS Transaction Server for z/OS, CICS for MVS/ESA™, CICS Transaction Server for VSE/ESA™, and CICS/VSE®.

CICS TS

Abbreviation of CICS Transaction Server.

class In object-oriented programming, a model or template that can be instantiated to create objects with a common definition and therefore, common properties, operations, and behavior. An object is an instance of a class.

classpath

In the execution environment, an environment variable keyword that specifies the directories in which to look for class and resource files.

Client API

The Client API is the interface used by Client applications to invoke services in CICS using the Client daemon. See External Call Interface, External Presentation Interface, and External Security Interface.

Client application

The client application is a user application written in a supported programming language, other than Java, that uses the Client API.

Client daemon

The Client daemon, process cclclnt, exists only on UNIX, Windows, and Linux. It manages network connections to CICS servers. It processes ECI, EPI, and ESI requests, sending and receiving the appropriate flows from the CICS server to satisfy the application requests. It uses the CLIENT section of ctg.ini for its configuration.

client/server

Pertaining to the model of interaction in distributed data processing in which a program on one computer sends a request to a program on another computer and awaits a response. The requesting program is called a client; the answering program is called a server.

CNOS See *Change-Number-of-Sessions*.

code page

An assignment of hexadecimal identifiers (code points) to graphic characters. Within a given code page, a code point can have only one meaning.

color mapping file

A file that is used to customize the 3270 screen color attributes on client workstations.

commit phase

The second phase in a XA process. If all participants acknowledge that they are prepared to commit, the transaction manager issues the commit request. If any participant is not prepared to commit the transaction manager issues a back-out request to all participants.

communication area (COMMAREA)

A communication area that is used for passing data both between programs within a transaction and between transactions.

configuration file

A file that specifies the characteristics of a program, system device, server or network.

connection

In data communication, an association established between functional units for conveying information.

In Open Systems Interconnection architecture, an association established by a given layer between two or more entities of the next higher layer for the purpose of data transfer.

In TCP/IP, the path between two protocol application that provides reliable data stream delivery service.

In Internet, a connection extends from a TCP application on one system to a TCP application on another system.

container

A container is a named block of data designed for passing information between programs. A container is a "named COMMAREA" that is not limited to 32KB. Containers are grouped together in sets called channels.

container-managed transaction

A transaction where the EJB container is responsible for administration of tasks such as committal or rollback. See also *bean-managed transaction*.

control table

In CICS, a storage area used to describe or define the configuration or operation of the system.

conversation

A connection between two programs over a session that allows them to communicate with each other while processing a transaction.

conversation security

In APPC, a process that allows validation of a user ID or group ID and password before establishing a connection.

daemon

A program that runs unattended to perform continuous or periodic systemwide functions, such as network control. A daemon may be launched automatically, such as when the operating system is started, or manually.

data link control (DLC)

A set of rules used by nodes on a data link (such as an SDLC link or a token ring) to accomplish an orderly exchange of information.

DBCS See *double-byte character set*.

dependent logical unit

A logical unit that requires assistance from a system services control point (SSCP) to instantiate an LU-to-LU session.

deprecated

Pertaining to an entity, such as a programming element or feature, that is supported but no longer recommended, and that might become obsolete.

digital certificate

An electronic document used to identify an individual, server, company, or some other entity, and to associate a public key with the entity. A digital certificate is issued by a certificate authority and is digitally signed by that authority.

digital signature

Information that is encrypted with an entity's private key and is appended to a message to assure the recipient of the authenticity and integrity of the message. The digital signature proves that the message was signed by the entity that owns, or has access to, the private key or shared secret symmetric key.

distributed application

An application for which the component application programs are distributed between two or more interconnected processors.

distributed processing

The processing of different parts of the same application in different systems, on one or more processors.

distributed program link (DPL)

A link that enables an application program running on one CICS system to link to another application program running in another CICS system.

DLL See *dynamic link library*.

domain

In the Internet, a part of a naming hierarchy in which the domain name consists of a sequence of names (labels) separated by periods (dots).

domain name

In TCP/IP, a name of a host system in a network.

domain name server

In TCP/IP, a server program that supplies name-to-address translation by mapping domain names to internet addresses. Synonymous with name server.

dotted decimal notation

The syntactical representation for a 32-bit integer that consists of four 8-bit numbers written in base 10 with periods (dots) separating them. It is used to represent IP addresses.

double-byte character set (DBCS)

A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. Contrast with *single-byte character set*.

DPL See *distributed program link*.

dynamic link library (DLL)

A collection of runtime routines made available to applications as required.

EBCDIC

See *Extended binary-coded decimal interchange code*.

ECI See *external call interface*.

EJB See *Enterprise JavaBeans*.

emulation program

A program that allows a host system to communicate with a workstation in the same way as it would with the emulated terminal.

emulator

A program that causes a computer to act as a workstation attached to another system.

encryption

The process of transforming data into an unintelligible form in such a way that the original data can be obtained only by using a decryption process.

enterprise bean

A Java component that can be combined with other resources to create J2EE applications. There are three types of enterprise beans: entity beans, session beans, and message-driven beans.

Enterprise JavaBeans

A component architecture defined by Sun Microsystems for the development and deployment of object-oriented, distributed, enterprise-level applications (J2EE).

environment variable

A variable that specifies the operating environment for a process. For example, environment variables can describe the home directory, the command search path, the terminal in use, and the current time zone.

EPI See *external presentation interface*.

ESI See *external security interface*.

Ethernet

A local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and transmission. Ethernet uses carrier sense multiple access with collision detection (CSMA/CD).

EXCI See *External CICS Interface*.

external call interface (ECI)

A facility that allows a non-CICS program to run a CICS program. Data is exchanged in a COMMAREA as for normal CICS interprogram communication.

Extended binary-coded decimal interchange code (EBCDIC)

A coded character set of 256 8-bit characters developed for the representation of textual data.

extended logical unit of work (extended LUW)

A logical unit of work that is extended across successive ECI requests to the same CICS server.

External CICS Interface (EXCI)

The EXCI is an MVS™ application programming interface provided by CICS Transaction Server for z/OS that enables a non-CICS program to call a CICS program and to pass and receive data using a COMMAREA or container. The CICS application program is invoked as if linked-to by another CICS application program.

external presentation interface (EPI)

A facility that allows a non-CICS program to appear to CICS as one or more standard 3270 terminals. 3270 data can be presented to the user by emulating a 3270 terminal or by using a graphical user interface.

external security interface (ESI)

A facility that enables client applications to verify and change passwords for user IDs on CICS servers.

firewall

A configuration of software that prevents unauthorized traffic between a trusted network and an untrusted network.

gateway

A device or program used to connect two systems or networks.

gateway classes

The Gateway Classes are the Java class library used by Java Client applications to invoke services in CICS.

Gateway daemon

The Gateway daemon is a long-running Java process used only in remote mode. The Gateway daemon listens for network requests from remote Java Client applications. It issues these requests to CICS using the CICS connectivity components. These are the Client daemon on UNIX, Windows, and Linux platforms, and EXCI or IPIC on z/OS. The Gateway daemon runs the protocol listener threads, the connection manager threads, and the worker threads. It uses the GATEWAY section of ctg.ini (and on z/OS the STDENV file or the ctgenvar script) for its configuration.

Gateway group

A collection of Gateway daemon instances, that uses the services of a single ctgmaster. The group provides a TCP/IP load balancing capability for XA transactions.

gateway token

Gateway tokens are used in the statistical data API. A token represents a specific Gateway daemon, once a connection is established successfully.

global transaction

A recoverable unit of work performed by one or more resource managers in a distributed transaction processing environment and coordinated by an external transaction manager.

host A computer that is connected to a network (such as the Internet or an SNA network) and provides an access point to that network. The host can be any system; it does not have to be a mainframe.

host address

An IP address that is used to identify a host on a network.

host ID

In TCP/IP, that part of the Internet address that defines the host on the network. The length of the host ID depends on the type of network or network class (A, B, or C).

host name

In the Internet suite of protocols, the name given to a computer. Sometimes, host name is used to mean the fully qualified domain name; other times, it is used to mean the most specific subname of a fully qualified domain name. For example, if mycomputer.city.company.com is the fully qualified domain name, either of the following may be considered the host name: mycomputer.city.company.com, mycomputer.

hover help

Information that can be viewed by holding a mouse over an item such as an icon in the user interface.

HTTP See *Hypertext Transfer Protocol*.

HTTPS

See *Hypertext Transfer Protocol Secure*.

Hypertext Transfer Protocol

In the Internet suite of protocols, the protocol that is used to transfer and display hypertext and XML documents.

Hypertext Transfer Protocol Secure

A TCP/IP protocol that is used by World Wide Web servers and Web browsers to transfer and display hypermedia documents securely across the Internet.

ID data

An ID data structure holds an individual result from a statistical API function.

iKeyman

A tool for maintaining digital certificates for JSSE.

independent logical unit

A logical unit (LU) that can both send and receive a BIND, and which supports single, parallel, and multiple sessions. See *BIND*.

Internet Architecture Board

The technical body that oversees the development of the internet suite of protocols known as TCP/IP.

Internet Protocol (IP)

In TCP/IP, a protocol that routes data from its source to its destination in an Internet environment.

interoperability

The capability to communicate, execute programs, or transfer data among various functional units in a way that requires the user to have little or no knowledge of the unique characteristics of those units.

IP Internet Protocol.

IPIC See IP interconnectivity (IPIC).

IP address

A unique address for a device or logical unit on a network that uses the IP standard.

IP interconnectivity (IPIC)

The IPIC protocol enables Distributed Program Link (DPL) access from a non-CICS program to a CICS program over TCP/IP, using the External Call Interface (ECI). IPIC passes and receives data using COMMAREAs, or containers.

J2EE See *Java 2 Platform Enterprise Edition*

J2EE Connector architecture (JCA)

A standard architecture for connecting the J2EE platform to heterogeneous enterprise information systems (EIS).

Java An object-oriented programming language for portable interpretive code that supports interaction among remote objects.

Java 2 Platform Enterprise Edition (J2EE)

An environment for developing and deploying enterprise applications, defined by Sun Microsystems Inc. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that allow multitiered, Web-based applications to be developed.

JavaBeans

As defined for Java by Sun Microsystems, a portable, platform-independent, reusable component model.

Java Client application

The Java client application is a user application written in Java, including servlets and enterprise beans, that uses the Gateway classes.

Java Development Kit (JDK)

The name of the software development kit that Sun Microsystems provided for the Java platform, up to and including v 1.1.x. Sometimes used erroneously to mean the Java platform or as a generic term for any software developer kits for Java.

JavaGateway

The URL of the CICS Transaction Gateway with which the Java Client application will communicate. The JavaGateway takes the form `protocol://address:port`. These protocols are supported: `tcp://`, `ssl://`, and `local:`. The CICS Transaction Gateway runs with the default port value of 2006. This parameter is not relevant if you are using the protocol `local:`. For example, you might specify a JavaGateway of `tcp://ctg.business.com:2006`. If you specify the protocol as `local:` you will connect directly to the CICS server, bypassing any CICS Transaction Gateway servers.

Java Native Interface (JNI)

A programming interface that allows Java code running in a Java virtual machine to work with functions that are written in other programming languages.

Java Runtime Environment (JRE)

A subset of the Java Software Development Kit (SDK) that supports the execution, but not the development, of Java applications. The JRE comprises the Java Virtual Machine (JVM), the core classes, and supporting files.

Java Secure Socket Extension (JSSE)

A Java package that enables secure Internet communications. It implements a Java version of the Secure Sockets Layer (SSL) and Transport Layer Security (TSL) protocols and supports data encryption, server authentication, message integrity, and optionally client authentication.

Java virtual machine (JVM)

A software implementation of a processor that runs compiled Java code (applets and applications).

JDK See *Java development kit (JDK)*.

JCA See *J2EE Connector Architecture (JCA)*.

- JNI** See *Java Native Interface (JNI)*.
- JRE** See *Java Runtime Environment*
- JSSE** See *Java Secure Socket Extension (JSSE)*.
- JVM** See *Java Virtual Machine (JVM)*.

keyboard mapping

A list that establishes a correspondence between keys on the keyboard and characters displayed on a display screen, or action taken by a program, when that key is pressed.

key ring

In the JSSE protocol, a file that contains public keys, private keys, trusted roots, and certificates.

local mode

“Local mode” describes the use of the CICS Transaction Gateway *local* protocol. The Gateway daemon is not used in local mode.

local transaction

A recoverable unit of work managed by a resource manager and not coordinated by an external transaction manager

logical unit (LU)

In SNA, a port through which an end user accesses the SNA network in order to communicate with another end user and through which the end user accesses the functions provided by system services control points (SSCP). An LU can support at least two sessions, one with an SSCP and one with another LU, and may be capable of supporting many sessions with other logical units. See *network addressable unit, primary logical unit, secondary logical unit*.

logical unit 6.2 (LU 6.2)

A type of logical unit that supports general communications between programs in a distributed processing environment.

The LU type that supports sessions between two applications using APPC.

logical unit of work (LUW)

A recoverable unit of work performed within CICS.

LU-LU session

In SNA, a session between two logical units (LUs) in an SNA network. It provides communication between two end users, or between an end user and an LU services component.

LU-LU session type 6.2

In SNA, a type of session for communication between peer systems. Synonymous with APPC protocol.

LUW See *logical unit of work*.

managed mode

Describes an environment in which connections are obtained from connection factories that the J2EE server has set up. Such connections are owned by the J2EE server.

medium access control (MAC) sublayer

One of two sublayers of the ISO Open Systems Interconnection data link layer proposed for local area networks by the IEEE Project 802 Committee on Local Area Networks and the European Computer Manufacturers Association (ECMA). It provides functions that depend on the topology of the network and uses services of the physical layer to provide services to the logical link control (LLC) sublayer. The OSI data link layer corresponds to the SNA data link control layer.

method

In object-oriented programming, an operation that an object can perform. An object can have many methods.

mode In SNA, a set of parameters that defines the characteristics of a session between two LUs.

name server

In TCP/IP, synonym for Domain Name Server. In Internet communications, a host that translates symbolic names assigned to networks and hosts into Internet addresses.

network address

In SNA, an address, consisting of subarea and element fields, that identifies a link, link station, or network addressable unit (NAU). Subarea nodes use network addresses; peripheral nodes use local addresses. The boundary function in the subarea node to which a peripheral node is attached transforms local addresses to network addresses and vice versa. See also *network name*.

network addressable unit (NAU)

In SNA, a logical unit, a physical unit, or a system services control point. The NAU is the origin or the destination of information transmitted by the path control network. See also *logical unit*, *network address*, *network name*.

network name

In SNA, the symbolic identifier by which end users refer to a network addressable unit (NAU), link station, or link. See also *network address*.

node type

In SNA, a designation of a node according to the protocols it supports and the network addressable units (NAUs) it can contain. Four types

are defined: 1, 2, 4, and 5. Type 1 and type 2 nodes are peripheral nodes; type 4 and type 5 nodes are subarea nodes.

nonmanaged mode

An environment in which the application is responsible for generating and configuring connection factories. The J2EE server does not own or know about these connection factories and therefore provides no Quality of Service facilities.

object In object-oriented programming, a concrete realization of a class that consists of data and the operations associated with that data.

object-oriented (OO)

Describing a computer system or programming language that supports objects.

one-phase commit

A protocol with a single commit phase, that is used for the coordination of changes to recoverable resources when a single resource manager is involved.

pacing

A technique by which a receiving station controls the rate of transmission of a sending station to prevent overrun.

parallel session

In SNA, two or more concurrently active sessions between the same two LUs using different pairs of network addresses. Each session can have independent session parameters.

PING In Internet communications, a program used in TCP/IP networks to test the ability to reach destinations by sending the destinations an Internet Control Message Protocol (ICMP) echo request and waiting for a reply.

partner logical unit (PLU)

In SNA, the remote participant in a session.

partner transaction program

The transaction program engaged in an APPC conversation with a local transaction program.

PLU See *primary logical unit* and *partner logical unit*.

port An endpoint for communication between devices, generally referring to a logical connection. A 16-bit number identifying a particular Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) resource within a given TCP/IP node.

prepare phase

The first phase of a XA process in which all participants are requested to confirm readiness to commit.

presentation logic

The part of a distributed application that is concerned with the user interface of the application. Compare with *business logic*.

primary logical unit (PLU)

In SNA, the logical unit that contains the primary half-session for a particular logical unit-to-logical unit (LU-to-LU) session. See also *secondary logical unit*.

protocol boundary

The signals and rules governing interactions between two components within a node.

Query strings

Query strings are used in the statistical data API. A query string is an input parameter, specifying the statistical data to be retrieved.

Resource Access Control Facility (RACF®)

An IBM licensed program that provides access control by identifying users to the system; verifying users of the system; authorizing access to protected resources; logging detected unauthorized attempts to enter the system; and logging detected accesses to protected resources.

region In workload management on CICS Transaction Gateway for Windows, an instance of a CICS server.

remote mode

“Remote mode” describes the use of one of the supported CICS Transaction Gateway network protocols to connect to the Gateway daemon.

remote procedure call (RPC)

A protocol that allows a program on a client computer to run a program on a server.

request unit (RU)

In SNA, a message unit that contains control information such as a request code, or function management (FM) headers, end-user data, or both.

request/response unit

A generic term for a request unit or a response unit. See also *request unit* and *response unit*.

response file

A file that contains predefined values that is used instead of someone having to enter those values one at a time. See *CID methodology*.

response unit (RU)

A message unit that acknowledges a request unit; it may contain prefix information received in a request unit.

resource group ID

A resource group ID is a logical grouping of resources, grouped for statistical purposes. A resource group ID is associated with a number of resource group statistics, each identified by a statistic ID.

resource ID

A resource ID refers to a specific resource. Information about the resource is included in resource-specific statistics. Each statistic is identified by a statistic ID.

resource manager

The participant in a transaction responsible for controlling access to recoverable resources. In terms of the CICS resource adapters this is represented by an instance of a ConnectionFactory.

Resource Recovery Services (RRS)

A z/OS facility that provides two-phase sync point support across participating resource managers.

Result set

A result set is a set of data calculated or recorded by a statistical API function.

Result set token

A result set token is a reference to the set of results returned by a statistical API function.

rollback

An operation in a transaction that reverses all the changes made during the unit of work. After the operation is complete, the unit of work is finished. Also known as a backout.

RU Request unit. Response unit.

RPC See *remote procedure call*.

SBCS See *single-byte character set*.

secondary logical unit (SLU)

In SNA, the logical unit (LU) that contains the secondary half-session for a particular LU-LU session. Contrast with primary logical unit. See also *logical unit*.

Secure Sockets Layer (SSL)

A security protocol that provides communication privacy. SSL enables client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery. SSL applies only to internet protocols, and is not applicable to SNA.

servlet

A Java program that runs on a Web server and extends the server's

functionality by generating dynamic content in response to Web client requests. Servlets are commonly used to connect databases to the Web.

session limit

In SNA, the maximum number of concurrently active logical unit to logical unit (LU-to-LU) sessions that a particular logical unit (LU) can support.

single-byte character set (SBCS)

A character set in which each character is represented by 1 byte. Contrast with double-byte character set.

sign-on capable terminal

A sign-on capable terminal allows sign-on transactions, either CICS-supplied (CESN) or user-written, to be run. Contrast with sign-on incapable terminal.

SIT See *system initialization table*.

SNA sense data

An SNA-defined encoding of error information. In SNA, the data sent with a negative response, indicating the reason for the response.

SNASVCMG mode name

The SNA service manager mode name. This is the architecturally-defined mode name identifying sessions on which CNOS is exchanged. Most APPC-providing products predefine SNASVCMG sessions.

socket A network communication concept, typically representing a point of connection between a client and a server. A TCP/IP socket will normally combine a host name or IP address, and a port number.

SSL See *Secure Sockets Layer (SSL)*.

SSLight

An implementation of SSL, written in Java, and no longer supported by CICS Transaction Gateway.

statistic data

A statistic data structure holds individual statistical result returned after calling a statistical API function.

statistic group

A statistic group is a generic term for a collection of statistic IDs.

statistic ID

A statistic ID is a label referring to a specific statistic. A statistic ID is used to retrieve specific statistical data, and always has a direct relationship with a statistic group.

system initialization table

A table containing parameters used to start a CICS control region.

System Management Interface Tool (SMIT)

An interface tool of the AIX operating system for installing, maintaining, configuring, and diagnosing tasks.

standard error

In many workstation-based operating systems, the output stream to which error messages or diagnostic messages are sent.

subnet

An interconnected, but independent segment of a network that is identified by its Internet Protocol (IP) address.

subnet address

In Internet communications, an extension to the basic IP addressing scheme where a portion of the host address is interpreted as the local network address.

sync point

A logical point in the execution of program where the changes made by the program are consistent and complete, and can be committed. The output, which has been held up to that point, is sent to its destination, the input is removed from the message queues, and updates are made available to other applications. When a program terminates abnormally, CICS recovery and restart facilities do not backout updates prior to the last completed sync point.

Systems Network Architecture (SNA)

An architecture that describes the logical structure, formats, protocols, and operational sequences for transmitting information units through the networks and also the operational sequences for controlling the configuration and operation of networks.

System SSL

An implementation of SSL, no longer supported by CICS Transaction Gateway on z/OS.

TCP62 SNA logical unit type 62 (LU62) protocol encapsulated in TCP/IP. This allows APPC applications to communicate over a TCP/IP Network without changes to the applications.

TCP/IP

See Transmission Control Protocol/Internet Protocol.

TCP/IP load balancing

The ability to distribute TCP/IP connections across target servers.

terminal emulation

The capability of a microcomputer or personal computer to operate as

if it were a particular type of terminal linked to a processing unit and to access data. See also *emulator, emulation program*.

thread A stream of computer instructions that is in control of a process. In some operating systems, a thread is the smallest unit of operation in a process. Several threads can run concurrently, performing different jobs.

timeout A time interval that is allotted for an event to occur or complete before operation is interrupted.

TLS See *Transport Layer Security (TLS)*.

token-ring network A local area network that connects devices in a ring topology and allows unidirectional data transmission between devices by a token-passing procedure. A device must receive a token before it can transmit data.

trace A record of the processing of a computer program. It exhibits the sequences in which the instructions were processed.

transaction manager A software unit that coordinates the activities of resource managers by managing global transactions and coordinating the decision to commit them or roll them back.

transaction program A program that uses the Advanced Program-to-Program Communications (APPC) application programming interface (API) to communicate with a partner application program on a remote system.

Transmission Control Protocol/Internet Protocol (TCP/IP) An industry-standard, nonproprietary set of communications protocols that provide reliable end-to-end connections between applications over interconnected networks of different types.

Transport Layer Security (TLS) A security protocol that provides communication privacy. TLS enables client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery. TLS applies only to internet protocols, and is not applicable to SNA. TLS is also known as SSL 3.1.

two-phase commit A protocol with both a prepare and a commit phase, that is used for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction.

type 2.0 node

A node that attaches to a subarea network as a peripheral node and provides a range of end-user services but no intermediate routing services.

type 2.1 node

An SNA node that can be configured as an endpoint or intermediate routing node in a network, or as a peripheral node attached to a subarea network.

Uniform Resource Locator (URL)

A sequence of characters that represent information resources on a computer or in a network such as the Internet. This sequence of characters includes (a) the abbreviated name of the protocol used to access the information resource and (b) the information used by the protocol to locate the information resource.

unit of recovery (UR)

A defined package of work to be performed by the RRS.

unit of work (UOW)

A recoverable sequence of operations performed by an application between two points of consistency. A unit of work begins when a transaction starts or at a user-requested sync point. It ends either at a user-requested sync point or at the end of a transaction.

user session

Any APPC session other than a SNASVCMG session.

verb

A reserved word that expresses an action to be taken by an application programming interface (API), a compiler, or an object program.

In SNA, the general name for a transaction program's request for communication services.

version string

A character string containing version information about the statistical data API.

Web browser

A software program that sends requests to a Web server and displays the information that the server returns.

Web server

A software program that responds to information requests generated by Web browsers.

wide area network (WAN)

A network that provides communication services to a geographic area

larger than that served by a local area network or a metropolitan area network, and that may use or provide public communication facilities.

wrapping trace

A configuration in which the **Maximum Client wrap size** setting is greater than 0. The total size of Client daemon binary trace files is limited to the value specified in the **Maximum Client wrap size** setting. With standard I/O tracing, two files, called `cicscli.bin` and `cicscli.wrp`, are used; each can be up to half the size of the **Maximum Client wrap size**.

XA requests

An XA request is any request sent or received by the CICS Transaction Gateway in support of an XA transaction. These requests include the XA commands `commit`, `complete`, `end`, `forget`, `prepare`, `recover`, `rollback`, and `start`.

XA transaction

A global transaction that adheres to the X/Open standard for distributed transaction processing (DTP.)

Index

Special characters

- _CICS_DIED return code
 - CICS_ChangePassword function 176
 - CICS_VerifyPassword function 173
- :=
 - in CclBuf class 54
 - in Public methods 54
- (parameter)
 - in changed 58
- <install_path> vii

A

- abendCode
 - in CclException class 67
 - in CclFlow class 75
 - in Public methods 67, 75
- AbendCode
 - in Methods
 - in Flow COM Class 23
- accessibility 243
- active
 - in state 66
 - in State 66
- activeFlow
 - in CclConn class 56
 - in CclFlow class 74
 - in CclUOW class 97
- activeUOW
 - in CclConn class 56
 - in CclUOW class 97
- AID
 - in CclScreen class 83
 - in Enumerations 83
- alphanumeric
 - in BaseType 73
 - in inputType 70
- alterSecurity
 - in CclConn class 57
 - in CclTerminal class 89
 - in Public methods 57, 89
- AlterSecurity
 - in Methods
 - in Connect COM Class 5
 - in Terminal COM Class 35
- alterSecurity (parameter)
 - in alterSecurity 57
- AppendString
 - in Methods
 - in Buffer COM Class 2

- appendText
 - in CclField class 68
 - in Public methods 68
- AppendText
 - in Methods
 - in Field COM Class 17
- array (parameter)
 - in SetData 3
- assign
 - in CclBuf class 51
 - in Public methods 51
- async
 - in CclSession constructor 86
 - in Sync 49
- ATISState
 - in CclTerminal class 95
 - in Enumerations 95
- ATISState parameter
 - CICS_EpiATISState function 164
- attachTran (parameter)
 - in CclConn constructor 56, 57
 - in TranDetails 9
- attribute (parameter)
 - in setBaseAttribute 71
 - in setExtAttribute 71
- Attribute (parameter)
 - in SetBaseAttribute 21
 - in SetExtAttribute 21
- Attributes parameter
 - CICS_EpiAddExTerminal function 153
- available
 - in ServerStatus 61

B

- backgroundColor
 - in CclField class 69
 - in Public methods 69
- BackgroundColor
 - in Methods
 - in Field COM Class 18
- backout
 - in CallType 78
 - in CclUOW class 97
 - in Public methods 97
- Backout
 - in Poll 24
- BackOut
 - in Methods
 - in UOW COM Class 45

- baseAttribute
 - in CclField class 69
 - in Public methods 69
- BaseAttribute
 - in Methods
 - in Field COM Class 18
- BaseInts
 - in CclField class 72
 - in Enumerations 72
- BaseMDT
 - in CclField class 72
 - in Enumerations 72
- BaseProt
 - in CclField class 73
 - in Enumerations 73
- BaseType
 - in CclField class 73
 - in Enumerations 73
- black
 - in Color 73
- blinkHlt
 - in Highlight 73
- blue
 - in Color 73
- books 239
- Bool
 - in Ccl class 49
 - in Enumerations 49
- Buffer
 - in AppendString 2
 - in Buffer COM Class 1
 - in Link 7
 - in Poll 24
 - in String 4
- buffer (parameter)
 - := 54
 - in CclBuf 51
 - r= 53
- Buffer COM class
 - Methods
 - AppendString 2
 - Data 2
 - ExtractString 2
 - InsertString 2
 - Length 3
 - Overlay 3
 - SetData 3
 - SetLength 3
 - SetString 4
 - String 4
- C**
- callback routine
 - ECl 109, 114, 119, 122
 - EPI 149, 152
- callType
 - in CclFlow class 75
 - in Public methods 75
- CallType
 - in CclFlow class 77
 - in Enumerations 77
 - in Methods
 - in Flow COM Class 23
- callTypeText
 - in CclFlow class 75
 - in Public methods 75
- CallTypeText
 - in Methods
 - in Flow COM Class 23
- cancel
 - in CallType 78
 - in CclConn class 57
 - in Public methods 57
- Cancel
 - in Methods
 - in Connect COM Class 5
 - in Poll 24
- Ccal Screen.fieldbyPosition method
 - in Field COM class 17
- Ccl class
 - Bool 49
 - Sync 49
- Ccl.Field
 - in FieldByName 27
- Ccl.Screen
 - in Send 41
- cclActive
 - in State 16, 34
- cclAlphanumeric
 - in InputType 20
- cclATIDisabled
 - in QueryATI 40
 - in SetATI 42
- cclATIEnabled
 - in QueryATI 40
 - in SetATI 41
- cclAvailable
 - in ServerStatus 8
- CclBuf
 - in CclBuf class 50, 51
 - in CclBuf constructors 50, 51
- CclBuf class
 - := 54
 - assign 51
 - CclBuf 50, 51
 - cut 52
 - dataArea 52
 - dataAreaLength 52
 - dataAreaOwner 52
 - DataAreaOwner 55

CclBuf class (*continued*)

- dataAreaType 52
- DataAreaType 55
- dataLength 53
- insert 53
- listState 53
- r= 53
- replace 55
- setDataLength 55

CclBuf constructors

- CclBuf 50, 51
- in CclBuf class 50

cclClient

- in State 34

CclConn class

- alterSecurity 57
- cancel 57
- change password 58
- changed 57
- link 58
- listState 59
- makeSecurityDefault 59
- password 59
- serverName 59, 60
- serverStatus 60
- ServerStatus 61
- serverStatusText 60
- status 60
- userId 60, 61
- verifyPassword 61

CclConn constructor

- in CclConn class 56

cclDark

- in Intensity 20

cclDiscon

- in State 16, 35

cclDSync

- in SetSyncType 24, 25, 34

CclECI class

- exCode 62
- exCodeText 62
- handleException 62
- instance 63
- listState 63
- serverCount 63
- serverDesc 63
- serverName 63

CclECI constructor (protected)

- in CclECI class 62

CclEPI class

- diagnose 64
- exCode 64
- exCodeText 64
- handleException 65
- serverCount 65

CclEPI class (*continued*)

- serverDesc 65
- serverName 65
- state 66
- State 66
- terminate 66

CclEPI constructor

- in CclEPI class 64

cclError

- in State 16, 35

CclException class

- abendCode 67
- className 67
- diagnose 67
- exCode 67
- exCodeText 67
- exObject 67
- methodName 68

CclField class

- appendText 68
- backgroundColor 69
- baseAttribute 69
- BaseInts 72
- BaseMDT 72
- BaseProt 73
- BaseType 73
- Color 73
- column 69
- dataTag 69
- foregroundColor 69
- highlight 69
- Highlight 73
- inputProt 70
- inputType 70
- intensity 70
- length 70
- position 70
- resetDataTag 71
- row 71
- setBaseAttribute 71
- setExtAttribute 71
- setText 71, 72
- text 72
- textLength 72
- transparency 72
- Transparency 73

CclFlow

- in CclFlow class 74

- in CclFlow constructor 74

CclFlow class

- abendCode 75
- callType 75
- CallType 77
- callTypeText 75
- CclFlow 74

CclFlow class (*continued*)

- connection 75
- diagnose 75
- flowId 75
- forceReset 75
- handleReply 76
- listState 76
- poll 76
- setTimeout 77
- syncType 77
- timeout 77
- uow 77
- wait 77

CclFlow constructor

- CclFlow 74
- in CclFlow class 74

cclIntense

- in Intensity 20

CclMap class

- exCode 78
- exCodeText 79
- field 79
- namedField 79
- validate 80

CclMap constructor

- in CclMap class 78

cclModified

- in DataTag 18

cclNoError

- in ExCode 15

cclNormal

- in Intensity 20

cclNumeric

- in InputType 20

CclOSecTime

- in Ccl SecAttr interface 31

cclProtect

- in InputProt 19

CclScreen class

- AID 83
- cursorCol 81
- cursorRow 81
- depth 81
- field 81
- fieldCount 81
- mapName 82
- mapSetName 82
- setAID 82
- setCursor 82
- width 83

cclServer

- in State 34

CclSession class

- diagnose 86
- handleReply 86

CclSession class (*continued*)

- state 86
- State 87
- terminal 86
- transID 86

CclSession constructor

- in CclSession class 85

cclSync

- in SetSyncType 24, 25, 34

cclSystemError

- in ExCode 15

CclTerminal class

- alterSecurity 89
- ATIState 95
- CCSid 90
- changePassword 89
- diagnose 90
- disconnect 90
- discReason 90
- EndTerminalReason 96
- exCode 91
- exCodeText 91
- install 91
- makeSecurityDefault 91
- netName 92
- password 92
- poll 92
- queryATI 92
- readTimeout 93
- receiveATI 93
- screen 93
- send 93, 94
- serverName 94
- setATI 94
- signonCapability 94
- signonType 95
- state 94
- State 96
- termID 95
- transID 95
- userId 95
- verifyPassword 95

CclTerminal constructor

- in CclTerminal class 87

cclUnavailable

- in ServerStatus 8

cclUnknown

- in ServerStatus 8

cclUnknownServer

- in ExCode 15

cclUnmodified

- in DataTag 18
- in ResetDataTag 21

cclUnprotect

- in InputProt 19

- CclUOW class
 - backout 97
 - commit 97
 - forceReset 97
 - listState 98
 - uowId 98
- CclUOW constructor
 - in CclUOW class 97
- CCSid
 - in CclTerminal class 90
 - in Public methods 90
- CCSid
 - in Methods
 - in Terminal COM class 36
- CCSid (parameter)
 - in CclTerminal constructor 88
 - in SetTermDefns 42
- change password
 - in CclConn class 58
 - in Public methods 58
- changed
 - in CallType 78
 - in CclConn class 57
 - in Public methods 57
 - in ServerStatus 8
 - in ServerStatusText 8
- Changed
 - in Cancel 5
 - in Changed 5
 - in Details 6
 - in Methods
 - in Connect COM Class 5
 - in Poll 24
- changePassword
 - in CclTerminal class 89
 - in Public methods 89
- ChangePassword
 - in Methods
 - in Connect COM Class 5
 - in Terminal COM Class 36
- CICS_ChangePassword function
 - definition 174
- CICS_ECI_DESCRIPTION_MAX 132
- CICS_ECI_SYSTEM_MAX 132
- CICS_EciDataReturnExit 191
- CICS_EciDataSendExit 190
- CICS_EciExternalCallExit1 186
- CICS_EciExternalCallExit2 187
- CICS_EciInitializeExit 184
- CICS_EciListSystems function 132
 - ECI_ERR_INVALID_DATA_LENGTH 133
 - ECI_ERR_MORE_SYSTEMS 132
 - ECI_ERR_NO_CICS 133
 - ECI_ERR_NO_SYSTEMS 133
 - ECI_ERR_SYSTEM_ERROR 133
- CICS_EciListSystems function (*continued*)
 - ECI_NO_ERROR 132
- CICS_EciSetProgramAliasExit 192
- CICS_EciSystem_t data structure
 - definition 132
 - use 132
- CICS_EciSystemIdExit 188
- CICS_EciTerminateExit 185
- CICS_EPI_ADD_TERM event
 - definition 166
- CICS_EPI_ATI_HOLD 163, 164
- CICS_EPI_ATI_ON 163, 164
- CICS_EPI_ATI_QUERY 164
- CICS_EPI_DESCRIPTION_MAX, 134
- CICS_EPI_DEVTYPE_MAX 148, 152
- CICS_EPI_END_FAILED 170
- CICS_EPI_END_OUTSERVICE 169
- CICS_EPI_END_SHUTDOWN 169
- CICS_EPI_END_SIGNOFF 169
- CICS_EPI_END_UNKNOWN 170
- CICS_EPI_ERR_ABENDED return code
 - CICS_EpiReply function 163
- CICS_EPI_ERR_ADDTYPE_INVALID return code
 - CICS_EpiAddExTerminal function 154
- CICS_EPI_ERR_ALREADY_INSTALLED return code
 - CICS_EpiAddExTerminal function 154
 - CICS_EpiAddTerminal function 150
- CICS_EPI_ERR_ATI_ACTIVE return code
 - CICS_EpiStartTran function 161
- CICS_EPI_ERR_ATI_STATE return code
 - CICS_EpiATIState function 164
- CICS_EPI_ERR_BAD_INDEX return code
 - CICS_EpiATIState function 164
 - CICS_EpiDelTerminal function 156
 - CICS_EpiGetEvent function 165
 - CICS_EpiInquireSystem function 155
 - CICS_EpiPurgeTerminal function 157
 - CICS_EpiReply function 162
 - CICS_EpiSetSecurity function 158
 - CICS_EpiStartTran function 161
- CICS_EPI_ERR_CCsid_INVALID return code
 - CICS_EpiAddExTerminal function 154
- CICS_EPI_ERR_FAILED return code
 - CICS_EpiAddExTerminal function 153
 - CICS_EpiAddTerminal function 149
 - CICS_EpiATIState function 164
 - CICS_EpiDelTerminal function 156
 - CICS_EpiGetEvent function 165
 - CICS_EpiInitialize function 144
 - CICS_EpiInquireSystem function 155
 - CICS_EpiListSystems function 146
 - CICS_EpiPurgeTerminal function 157
 - CICS_EpiReply function 163
 - CICS_EpiStartTran function 161
 - CICS_EpiTerminate function 145

CICS_EPI_ERR_IN_CALLBACK return code
 CICS_EpiAddExTerminal function 153
 CICS_EpiAddTerminal function 149
 CICS_EpiATISState function 164
 CICS_EpiDelTerminal function 156
 CICS_EpiGetEvent function 166
 CICS_EpiInquireSystem function 155
 CICS_EpiListSystems function 146
 CICS_EpiPurgeTerminal function 157
 CICS_EpiReply function 163
 CICS_EpiSetSecurity function 159
 CICS_EpiStartTran function 161
 CICS_EpiTerminate function 145

CICS_EPI_ERR_IS_INIT return code
 CICS_EpiInitialize function 144

CICS_EPI_ERR_MAX_SESSIONS return code
 CICS_EpiAddExTerminal function 154
 CICS_EpiAddTerminal function 150
 CICS_EpiStartTran function 162

CICS_EPI_ERR_MAX_SYSTEMS return code
 CICS_EpiAddExTerminal function 154
 CICS_EpiAddTerminal function 150

CICS_EPI_ERR_MODEL_INVALID return code
 CICS_EpiAddExTerminal function 154
 CICS_EpiAddTerminal function 150

CICS_EPI_ERR_MORE_DATA return code
 CICS_EpiGetEvent function 166

CICS_EPI_ERR_MORE_EVENTS return code
 CICS_EpiGetEvent function 166

CICS_EPI_ERR_MORE_SYSTEMS return code
 CICS_EpiListSystems function 146

CICS_EPI_ERR_NO_CONVERSE return code
 CICS_EpiReply function 163

CICS_EPI_ERR_NO_DATA return code
 CICS_EpiReply function 163
 CICS_EpiStartTran function 161

CICS_EPI_ERR_NO_EVENT return code
 CICS_EpiGetEvent function 166

CICS_EPI_ERR_NO_SYSTEMS return code
 CICS_EpiListSystems function 146

CICS_EPI_ERR_NOT_3270_DEVICE return code
 CICS_EpiAddExTerminal function 154
 CICS_EpiAddTerminal function 150

CICS_EPI_ERR_NOT_INIT return code
 CICS_EpiAddExTerminal function 153
 CICS_EpiAddTerminal function 149
 CICS_EpiATISState function 164
 CICS_EpiDelTerminal function 156
 CICS_EpiGetEvent function 166
 CICS_EpiInquireSystem function 155
 CICS_EpiListSystems function 146
 CICS_EpiPurgeTerminal function 157
 CICS_EpiReply function 163
 CICS_EpiSetSecurity function 159
 CICS_EpiStartTran function 161

CICS_EPI_ERR_NOT_INIT return code (*continued*)
 CICS_EpiTerminate function 145

CICS_EPI_ERR_NULL_PARM return code
 CICS_EpiAddExTerminal function 153
 CICS_EpiAddTerminal function 149
 CICS_EpiGetEvent function 166
 CICS_EpiInquireSystem function 155
 CICS_EpiListSystems function 146

CICS_EPI_ERR_NULL_PASSWORD return code
 CICS_EpiSetSecurity function 159

CICS_EPI_ERR_NULL_USERID return code
 CICS_EpiSetSecurity function 159

CICS_EPI_ERR_PASSWORD_INVALID return code
 CICS_EpiAddExTerminal function 153
 CICS_EpiSetSecurity function 159

CICS_EPI_ERR_RESOURCE_SHORTAGE return code
 CICS_EpiAddExTerminal function 154
 CICS_EpiAddTerminal function 150
 CICS_EpiStartTran function 161

CICS_EPI_ERR_RESPONSE_TIMEOUT return code
 CICS_EpiAddExTerminal function 153

CICS_EPI_ERR_SECURITY return code
 CICS_EpiAddExTerminal function 153
 CICS_EpiAddTerminal function 149

CICS_EPI_ERR_SERVER_BUSY return code
 CICS_EpiAddExTerminal function 154
 CICS_EpiAddTerminal function 150

CICS_EPI_ERR_SERVER_DOWN return code
 CICS_EpiAddExTerminal function 153
 CICS_EpiAddTerminal function 149
 CICS_EpiReply function 163
 CICS_EpiStartTran function 161

CICS_EPI_ERR_SIGNON_NOT_POSS return code
 CICS_EpiAddExTerminal function 153

CICS_EPI_ERR_SIGNONCAP_INVALID return code
 CICS_EpiAddExTerminal function 154

CICS_EPI_ERR_SYSTEM return code
 CICS_EpiAddExTerminal function 153
 CICS_EpiAddTerminal function 149

CICS_EPI_ERR_SYSTEM_ERROR return code
 CICS_EpiSetSecurity function 159

CICS_EPI_ERR_TERMID_INVALID return code
 CICS_EpiAddExTerminal function 154
 CICS_EpiAddTerminal function 150

CICS_EPI_ERR_TRAN_ACTIVE return code
 CICS_EpiDelTerminal function 156

CICS_EPI_ERR_TTI_ACTIVE return code
 CICS_EpiStartTran function 161
 CICS_EpiTerminate function 145

CICS_EPI_ERR_USERID_INVALID return code
 CICS_EpiAddExTerminal function 154
 CICS_EpiSetSecurity function 159

CICS_EPI_ERR_VERSION return code
 CICS_EpiAddExTerminal function 154
 CICS_EpiInitialize function 144

CICS_EPI_ERR_VERSION return code *(continued)*
 CICS_EpiPurgeTerminal function 157
 CICS_EpiSetSecurity function 159
 CICS_EPI_ERR_WAIT return code
 CICS_EpiGetEvent function 166
 CICS_EPI_EVENT_CONVERSE event
 definition 167
 use 162, 167
 CICS_EPI_EVENT_END_TERM event
 definition 169
 use 138, 156, 157
 CICS_EPI_EVENT_END_TRAN event
 definition 168
 use 138, 159, 160
 CICS_EPI_EVENT_SEND event
 definition 167
 CICS_EPI_EVENT_START_ATI event
 definition 169
 use 161
 CICS_EPI_NETNAME_MAX 136, 147, 151
 CICS_EPI_NORMAL return code
 CICS_EpiAddExTerminal function 155
 CICS_EpiAddTerminal function 150
 CICS_EpiATISState function 164
 CICS_EpiDelTerminal function 156
 CICS_EpiGetEvent function 166
 CICS_EpiInitialize function 144
 CICS_EpiInquireSystem function 155
 CICS_EpiListSystems function 146
 CICS_EpiPurgeTerminal function 157
 CICS_EpiReply function 163
 CICS_EpiSetSecurity function 159
 CICS_EpiStartTran function 162
 CICS_EpiTerminate function 145
 CICS_EPI_NOWAIT 165
 CICS_EPI_NULL_PARAM return code
 CICS_EpiATISState function 164
 CICS_EPI_READTIMEOUT_EXPIRED 168
 CICS_EPI_SYSTEM_MAX 134, 147, 151
 CICS_EPI_TERM_INDEX_NONE 165
 CICS_EPI_TRAN_NO_ERROR 168
 CICS_EPI_TRAN_NOT_STARTED 168
 CICS_EPI_TRAN_STATE_UNKNOWN 168
 CICS_EPI_TRANSID_MAX 138, 160
 CICS_EPI_VERSION_200 144
 CICS_EPI_WAIT 165
 CICS_EpiAddExTerminal function
 definition 150
 use 134, 135, 136
 CICS_EpiAddTerminal function
 definition 147
 use 134, 136, 159, 163
 CICS_EpiAddTerminalExit 197
 CICS_EpiATISState function
 definition 163
 CICS_EpiAttributes_t data structure
 definition 134
 CICS_EpiDelTerminal function
 definition 156
 use 144, 156, 169
 CICS_EpiDelTerminalExit 205
 CICS_EpiDetails_t data structure
 definition 136
 use 147, 149, 150, 152
 CICS_EpiEventData_t data structure
 definition 138
 use 165
 CICS_EpiGetEvent function
 definition 164
 use 138, 139, 156, 161
 CICS_EpiGetEventExit 206
 CICS_EpiInitialize function
 definition 143
 use 144, 145
 CICS_EpiInitializeExit 195
 CICS_EpiInquireSystem function
 definition 155
 CICS_EpiListSystems function
 definition 145
 use 134
 CICS_EpiPurgeTerminal function
 ATI request cancellation 157
 definition 157
 CICS_EpiReply function
 definition 162
 use 167
 CICS_EpiReplyExit 204
 CICS_EpiSetSecurity function
 definition 158
 CICS_EpiStartTran function
 definition 159
 use 161, 168
 CICS_EpiStartTranExit 203
 CICS_EpiStartTranExtendedExit 202
 CICS_EpiSystem_t data structure
 definition 134
 use 146
 CICS_EpiSystemIdExit 208
 CICS_EpiTermIdExit 200
 CICS_EpiTermIdInfoExit 201
 CICS_EpiTerminate function
 definition 144
 CICS_EpiTerminateExit 196
 CICS_EpiTranFailedExit 210
 CICS_ESI_ERR_CALL_FROM_CALLBACK return code
 CICS_ChangePassword function 176
 CICS_SetDefaultSecurity function 178
 CICS_VerifyPassword function 173
 CICS_ESI_ERR_MAX_SESSIONS return code
 CICS_ChangePassword function 176

CICS_ESI_ERR_MAX_SESSIONS return code
(continued)
 CICS_VerifyPassword function 174

CICS_ESI_ERR_MAX_SYSTEMS return code
 CICS_ChangePassword function 176
 CICS_VerifyPassword function 174

CICS_ESI_ERR_NO_CICS return code
 CICS_ChangePassword function 176
 CICS_SetDefaultSecurity function 179
 CICS_VerifyPassword function 173

CICS_ESI_ERR_NO_SESSIONS return code
 CICS_ChangePassword function 176
 CICS_VerifyPassword function 173

CICS_ESI_ERR_NULL_NEW_PASSWORD return code
 CICS_ChangePassword function 177

CICS_ESI_ERR_NULL_OLD_PASSWORD return code
 CICS_ChangePassword function 177

CICS_ESI_ERR_NULL_PASSWORD return code
 CICS_VerifyPassword function 174

CICS_ESI_ERR_NULL_USERID return code
 CICS_ChangePassword function 176
 CICS_VerifyPassword function 174

CICS_ESI_ERR_PASSWORD_REJECTED return code
 CICS_ChangePassword function 177

CICS_ESI_ERR_PASSWORD_EXPIRED return code
 CICS_VerifyPassword function 174

CICS_ESI_ERR_PASSWORD_INVALID return code
 CICS_ChangePassword function 177
 CICS_SetDefaultSecurity function 179
 CICS_VerifyPassword function 174

CICS_ESI_ERR_PEM_NOT_SUPPORTED return code
 CICS_ChangePassword function 177
 CICS_VerifyPassword function 174

CICS_ESI_ERR_PEM_NOT_ACTIVE return code
 CICS_ChangePassword function 177
 CICS_VerifyPassword function 174

CICS_ESI_ERR_RESOURCE_SHORTAGE return code
 CICS_ChangePassword function 176
 CICS_VerifyPassword function 173

CICS_ESI_ERR_SECURITY_ERROR return code
 CICS_ChangePassword function 177
 CICS_VerifyPassword function 174

CICS_ESI_ERR_SYSTEM_ERROR return code
 CICS_ChangePassword function 176
 CICS_SetDefaultSecurity function 179
 CICS_VerifyPassword function 173

CICS_ESI_ERR_UNKNOWN_SERVER return code
 CICS_ChangePassword function 176
 CICS_SetDefaultSecurity function 179
 CICS_VerifyPassword function 173

CICS_ESI_ERR_USERID_INVALID return code
 CICS_ChangePassword function 177
 CICS_SetDefaultSecurity function 179
 CICS_VerifyPassword function 174

CICS_ESI_NO_ERROR return code
 CICS_ChangePassword function 176
 CICS_SetDefaultSecurity function 178
 CICS_VerifyPassword function 173

CICS_EsiDate_t data structure
 definition 170

CICS_EsiDetails_t data structure
 definition 171

CICS_EsiTime_t data structure
 definition 171

CICS_EXIT_BAD_ANCHOR return code
 CICS_EciDataReturnExit 192
 CICS_EciDataSendExit 191
 CICS_EciExternalCallExit1 186
 CICS_EciExternalCallExit2 188
 CICS_EciSetProgramAliasExit 193
 CICS_EciSystemIdExit 190
 CICS_EciTerminateExit 185
 CICS_EpiAddTerminalExit 198
 CICS_EpiDelTerminalExit 206
 CICS_EpiGetEventExit 207
 CICS_EpiReplyExit 205
 CICS_EpiStartTranExit 204
 CICS_EpiStartTranExtendedExit 203
 CICS_EpiSystemIdExit 209
 CICS_EpiTermIdExit 200
 CICS_EpiTermIdInfoExit 202
 CICS_EpiTerminateExit 197
 CICS_EpiTranFailedExit 211

CICS_EXIT_BAD_PARM return code
 CICS_EciDataReturnExit 192
 CICS_EciDataSendExit 191
 CICS_EciExternalCallExit1 186
 CICS_EciExternalCallExit2 188
 CICS_EciSetProgramAliasExit 193
 CICS_EciSystemIdExit 190
 CICS_EpiAddTerminalExit 199
 CICS_EpiDelTerminalExit 206
 CICS_EpiGetEventExit 207
 CICS_EpiReplyExit 205
 CICS_EpiStartTranExit 204
 CICS_EpiStartTranExtendedExit 203
 CICS_EpiSystemIdExit 209
 CICS_EpiTermIdExit 201
 CICS_EpiTermIdInfoExit 202
 CICS_EpiTranFailedExit 211

CICS_EXIT_BAD_STORAGE return code
 CICS_EciTerminateExit 185
 CICS_EpiTerminateExit 197

CICS_EXIT_CANT_INIT_EXITS return code
 CICS_EciInitializeExit 184
 CICS_EpiInitializeExit 196

CICS_EXIT_DONT_ADD_TERMINAL return code
 CICS_EpiAddTerminalExit 198
 CICS_EpiSystemIdExit 209

- CICS_EXIT_GIVE_UP return code
 - CICS_EciSystemIdExit 190
- CICS_EXIT_NO_EXIT return code
 - CICS_EciInitializeExit 184
 - CICS_EpiInitializeExit 196
- CICS_EXIT_OK return code
 - CICS_EciDataReturnExit 192
 - CICS_EciDataSendExit 191
 - CICS_EciExternalCallExit1 186
 - CICS_EciExternalCallExit2 188
 - CICS_EciInitializeExit 184
 - CICS_EciSetProgramAliasExit 193
 - CICS_EciSystemIdExit 189
 - CICS_EciTerminateExit 185
 - CICS_EpiAddTerminalExit 198
 - CICS_EpiDelTerminalExit 206
 - CICS_EpiGetEventExit 207
 - CICS_EpiInitializeExit 196
 - CICS_EpiReplyExit 205
 - CICS_EpiStartTranExit 204
 - CICS_EpiStartTranExtendedExit 203
 - CICS_EpiSystemIdExit 209
 - CICS_EpiTermIdExit 200
 - CICS_EpiTermIdInfoExit 201
 - CICS_EpiTerminateExit 197
 - CICS_EpiTranFailedExit 211
- CICS_ExternalCall 132
- CICS_SetDefaultSecurity function
 - definition 177
- CICS_VerifyPassword function
 - definition 172
- CicsClientStatus 131
- CicsServerStatus 131
- className
 - in CclException class 67
 - in Public methods 67
- clear
 - in AID 83
- client
 - in send 94
 - in State 87, 96
- code page 104, 111, 124, 128
- col
 - in validate 80
- col (parameter)
 - in setCurs 82
- Color
 - in CclField class 73
 - in Enumerations 73
- colPos (parameter)
 - in FieldByPosition 29
 - in SetCursor 30
- column
 - in CclField class 69
 - in Public methods 69

- Column
 - in Methods
 - in Field COM Class 18
- column (parameter)
 - in field 79, 81
- commarea (parameter)
 - in handleReply 76
 - in link 58, 59
 - in poll 76
- commArea (parameter)
 - in AbendCode 23
 - in Link 7
 - in Poll 24
- commit
 - in CallType 78
 - in CclUOW class 97
 - in Public methods 97
- Commit
 - in Methods
 - in UOW COM Class 46
 - in Poll 24
- Connect
 - in Connect COM Class 4
 - in Methods
 - in Terminal COM Class 36
 - in ServerName 8
 - in Terminal COM class 35
 - in UOW COM Class 45
 - in UserId 10
- Connect COM class
 - Methods
 - AlterSecurity 5
 - Cancel 5
 - Changed 5
 - ChangePassword 5
 - Details 6
 - Link 6
 - MakeSecurityDefault 7
 - Password 7
 - ServerName 8
 - ServerStatus 8
 - ServerStatusText 8
 - Status 8
 - TranDetails 9
 - UnpaddedPassword 9
 - UnpaddedServerName 9
 - UnpaddedUserid 10
 - UserId 10
 - VerifyPassword 10
- connection
 - in CclFlow class 75
 - in Public methods 75
- ConnectionType 131
- CreateObject 1

- cursorCol
 - in CclScreen class 81
 - in Public methods 81
- CursorCol
 - in Methods
 - in Screen COM Class 28
- cursorRow
 - in CclScreen class 81
 - in Public methods 81
- CursorRow
 - in Methods
 - in Screen COM Class 28
- cut
 - in CclBuf class 52
 - in Public methods 52
- cyan
 - in Color 73
- D**
- dark
 - in BaseInts 72
 - in intensity 70
- darkBlue
 - in Color 73
- Data
 - in Methods
 - in Buffer COM Class 2
- data conversion 104, 111, 124, 128
- Data parameter
 - CICS_EpiReply function 162
 - CICS_EpiStartTran function 160
- dataArea
 - in CclBuf class 52
 - in Public methods 52
- dataArea (parameter)
 - in assign 51
 - in CclBuf 51
 - in insert 53
 - in replace 55
- dataAreaLength
 - in CclBuf class 52
 - in Public methods 52
- dataAreaOwner
 - in CclBuf class 52
 - in Public methods 52
- DataAreaOwner
 - in CclBuf class 55
 - in Enumerations 55
- dataAreaType
 - in CclBuf class 52
 - in Public methods 52
- DataAreaType
 - in CclBuf class 55
 - in Enumerations 55
- dataLength
 - in CclBuf class 53
 - in link 59
 - in Public methods 53
- dataStream
 - in CclScreen class 80
- dataTag
 - in CclField class 69
 - in Public methods 69
- DataTag
 - in Methods
 - in Field COM Class 18
- Day
 - in SecTime COM Class 32
- default installation location vii
- defaultColor
 - in Color 73
- defaultHlt
 - in Highlight 73
- defaultTran
 - in Transparency 73
- depth
 - in CclScreen class 81
 - in Public methods 81
 - in validate 80
- Depth
 - in Methods
 - in Screen COM Class 28
- Description
 - CICS_EciListSystems 132
- Details
 - in Connect COM class 4
 - in Methods
 - in Connect COM Class 6
- Details parameter
 - CICS_ChangePassword function 176
 - CICS_EpiAddExTerminal function 152
 - CICS_EpiAddTerminal function 149
 - CICS_VerifyPassword function 173
- Devtype
 - in Methods
 - in Terminal COM Class 37
- devtype (parameter)
 - in CclTerminal constructor 87, 88
- devType (parameter)
 - in Connect 36
 - in SetTermDefns 42
- DevType parameter
 - CICS_EpiAddExTerminal function 152
 - CICS_EpiAddTerminal function 148
- DFHCNV macro 104, 111, 124, 128
- diagnose
 - in CclEPI class 64
 - in CclException class 67
 - in CclFlow class 75

- diagnose (*continued*)
 - in CclSession class 86
 - in CclTerminal class 90
 - in Public methods 64, 67, 75, 86, 90
- Diagnose
 - in EPI COM Class 13
 - in Methods
 - in EPI COM Class 14
 - in Flow COM Class 23
 - in Session COM Class 34
 - in Terminal COM Class 37
 - in State 16, 35
- disability 243
- disabled
 - in ATISState 95
 - in queryATI 93
 - in setATI 94
- discon
 - in state 66
 - in State 66, 87, 96
- disconnect
 - in CclTerminal class 90
 - in Public methods 90
- Disconnect
 - in Disconnect 37
 - in Methods
 - in Terminal COM Class 37
- DisconnectWithPurge
 - in Methods
 - in Terminal COM Class 37
- discReason
 - in CclTerminal class 90
 - in Public methods 90
- DiscReason
 - in DiscReason 37
 - in Methods
 - in Terminal COM Class 37
- display (parameter)
 - in ErrorWindow 11, 14
- documentation 239
- dsync
 - in CclSession constructor 86
 - in Sync 49

E

- ECI COM class
 - Methods
 - ErrorFormat 11
 - ErrorOffset 11
 - ErrorWindow 11
 - ExCode 12
 - ExCodeText 12
 - ServerCount 12
 - ServerDesc 12
 - ServerName 13

- ECI COM class (*continued*)
 - Methods (*continued*)
 - SetErrorFormat 13
- ECI exits 182
- ECI parameter block 99
- ECI status block 131
- eci_abend_code
 - field in ECI parameter block 99
 - with ECI_SYNC call type 103
- ECI_ASYNC call type
 - definition 108
- ECI_BACKOUT 102, 105, 110, 112
- eci_call_type 99
 - field in ECI parameter block 99
 - with ECL_ASYNC call type 109
 - with ECL_GET_REPLY call type 123
 - with ECL_GET_REPLY_WAIT call type 126
 - with ECL_GET_SPECIFIC_REPLY call type 127
 - with ECL_GET_SPECIFIC_REPLY_WAIT call type 130
 - with ECL_STATE_ASYNC call type 119
 - with ECL_STATE_SYNC call type 116
 - with ECL_SYNC call type 102
- eci_callback 109, 119
 - field in ECI parameter block 100
 - with ECL_ASYNC call type 114
 - with ECL_STATE_ASYNC call type 122
- ECL_CLIENTSTATE_INAPPLICABLE 131
- ECL_CLIENTSTATE_UNKNOWN 131
- ECL_CLIENTSTATE_UP 131
- eci_commarea
 - field in ECI parameter block 99
 - with ECL_ASYNC call type 111
 - with ECL_GET_REPLY call type 123
 - with ECL_GET_SPECIFIC_REPLY call type 127
 - with ECL_STATE_ASYNC call type 120
 - with ECL_STATE_SYNC call type 116
 - with ECL_SYNC call type 104
- eci_commarea_length
 - field in ECI parameter block 100
 - with ECL_ASYNC call type 111
 - with ECL_GET_REPLY call type 124
 - with ECL_GET_SPECIFIC_REPLY call type 128
 - with ECL_STATE_ASYNC call type 120
 - with ECL_STATE_SYNC call type 116
 - with ECL_SYNC call type 104
- ECL_COMMIT 102, 105, 110, 112
- ECL_CONNECTED_NOWHERE 131
- ECL_CONNECTED_TO_CLIENT 131
- ECL_CONNECTED_TO_SEVER 131
- ECL_ERR_ALREADY_ACTIVE 108, 115
- ECL_ERR_CALL_FROM_CALLBACK 101, 133
- ECL_ERR_CICS_DIED 107, 125, 129
- ECL_ERR_INVALID_CALL_TYPE 101

ECI_ERR_INVALID_DATA_LENGTH 107, 115, 118,
 122, 124, 128, 133
 ECI_ERR_INVALID_DATA_AREA 108, 115, 118, 122,
 125, 129
 ECI_ERR_INVALID_EXTEND_MODE 107, 115, 118,
 122
 ECI_ERR_INVALID_VERSION 101
 ECI_ERR_LUW_TOKEN 108, 115, 118, 122
 ECI_ERR_MAX_SESSIONS 108, 126, 130
 ECI_ERR_MAX_SYSTEMS 108, 126, 130
 ECI_ERR_MORE_SYSTEMS 132
 ECI_ERR_NO_CICS 107, 115, 125, 129, 133
 ECI_ERR_NO_REPLY 125, 129
 ECI_ERR_NO_SESSIONS 108, 115
 ECI_ERR_NO_SYSTEMS 133
 ECI_ERR_REQUEST_TIMEOUT 101
 ECI_ERR_RESOURCE_SHORTAGE 108, 115, 125, 129
 ECI_ERR_RESPONSE_TIMEOUT 101
 ECI_ERR_ROLLEDBACK 108, 125, 129
 ECI_ERR_SECURITY_ERROR 108, 126, 130
 ECI_ERR_SYSTEM_ERROR 101, 133
 ECI_ERR_THREAD_CREATE_ERROR 115, 125, 129
 ECI_ERR_TRANSACTION_ABEND 107, 125
 ECI_ERR_TRANSACTION_ABEND 129
 ECI_ERR_UNKNOWN_SERVER 108, 118, 126, 130
 eci_extend_mode 102, 103, 107, 110, 111, 114, 116, 117,
 120, 121, 123, 127
 field in ECI parameter block 100
 with ECI_ASYNC call type 112
 with ECI_STATE_ASYNC call type 120
 with ECI_STATE_SYNC call type 117
 with ECI_SYNC call type 104
 ECI_EXTENDED 105, 112
 ECI_GET_REPLY call type
 definition 123
 ECI_GET_REPLY_WAIT call type
 definition 126
 ECI_GET_SPECIFIC_REPLY call type
 definition 127
 ECI_GET_SPECIFIC_REPLY_WAIT call type
 definition 130
 eci_luw_token
 field in ECI parameter block 100
 with ECI_ASYNC call type 113
 with ECI_STATE_ASYNC call type 121
 with ECI_STATE_SYNC call type 117
 with ECI_SYNC call type 105
 eci_message_qualifier
 field in ECI parameter block 100
 with ECI_ASYNC call type 109, 112
 with ECI_GET_SPECIFIC_REPLY call type 127, 128
 with ECI_STATE_ASYNC call type 119, 121
 ECI_NO_ERROR 107, 115, 118, 122, 124, 128, 132
 ECI_NO_EXTEND 104, 112
 eci_password 103, 106, 110, 114
 eci_password (*continued*)
 field in ECI parameter block 99
 with ECI_ASYNC call type 110
 with ECI_SYNC call type 103
 eci_password2 103, 110
 field in ECI parameter block 101
 with ECI_ASYNC call type 114
 with ECI_SYNC call type 106
 eci_program_name
 field in ECI parameter block 99
 with ECI_ASYNC call type 110
 with ECI_SYNC call type 102
 ECI_SERVERSTATE_DOWN 131
 ECI_SERVERSTATE_UNKNOWN 131
 ECI_SERVERSTATE_UP 131
 ECI_STATE_ASYNC call type
 definition 118
 ECI_STATE_CANCEL 116, 117, 120, 121, 123, 127
 ECI_STATE_CHANGED 117, 121
 ECI_STATE_IMMEDIATE 117, 120
 ECI_STATE_SYNC call type
 definition 115
 ECI_STATUS 131
 ECI_SYNC call type
 definition 102
 eci_sysid
 field in ECI parameter block 100
 with ECI_ASYNC call type 113
 with ECI_GET_REPLY call type 124
 with ECI_GET_SPECIFIC_REPLY call type 128
 with ECI_STATE_ASYNC call type 121
 with ECI_STATE_SYNC call type 117
 with ECI_SYNC call type 106
 eci_system_name
 field in ECI parameter block 100
 with ECI_ASYNC call type 113
 with ECI_STATE_ASYNC call type 122
 with ECI_STATE_SYNC call type 118
 with ECI_SYNC call type 106
 eci_timeout
 field in ECI parameter block 100
 with ECI_SYNC call type 104, 111
 eci_tpn
 field in ECI parameter block 101
 with ECI_ASYNC call type 114
 with ECI_SYNC call type 107
 eci_transid
 field in ECI parameter block 99
 with ECI_ASYNC call type 110
 with ECI_SYNC call type 103
 eci_userid 103, 106, 110, 114
 field in ECI parameter block 99
 with ECI_ASYNC call type 110
 with ECI_SYNC call type 102
 eci_userid2 103, 110

- eci_userid2 (*continued*)
 - field in ECI parameter block 101
 - with ECI_ASYNC call type 114
 - with ECI_SYNC call type 106
- eci_version
 - field in ECI parameter block 100
 - with ECI_ASYNC call type 113
 - with ECI_GET_REPLY call type 124
 - with ECI_GET_SPECIFIC_REPLY call type 128
 - with ECI_STATE_ASYNC call type 121
 - with ECI_STATE_SYNC call type 118
 - with ECI_SYNC call type 106
- enabled
 - in ATISState 95
 - in queryATI 93
 - in setATI 94
- EndTerminalReason
 - in CclTerminal class 96
 - in Enumerations 96
- enter
 - in AID 83
- Enumerations
 - AID 83
 - ATISState 95
 - BaseInts 72
 - BaseMDT 72
 - BaseProt 73
 - BaseType 73
 - Bool 49
 - CallType 77
 - Color 73
 - DataAreaOwner 55
 - DataAreaType 55
 - EndTerminalReason 96
 - Highlight 73
 - in Ccl class 49
 - in CclBuf class 55
 - in CclConn class 61
 - in CclEPI class 66
 - in CclField class 72
 - in CclFlow class 77
 - in CclScreen class 83
 - in CclSession class 87
 - in CclTerminal class 95
 - ServerStatus 61
 - signonType 95
 - State 66, 87, 96
 - Sync 49
 - Transparency 73
- EPI
 - constants 133
 - data structures 134
 - events 166
 - functions 139
 - in EPI COM Class 13
- EPI COM class
 - Methods
 - Diagnose 14
 - ErrorFormat 14
 - ErrorOffset 14
 - ErrorWindow 14
 - ExCode 15
 - ExCodeText 15
 - ServerCount 15
 - ServerDesc 15
 - ServerName 16
 - SetErrorFormat 16
 - State 16
 - Terminate 17
- EPI exits 193
- error
 - in state 66
 - in State 66, 87, 96
- ErrorFormat
 - in Methods
 - in ECI COM Class 11
 - in EPI COM Class 14
- ErrorOffset
 - in Methods
 - in ECI COM Class 11
 - in EPI COM Class 14
- ErrorWindow
 - in Methods
 - in ECI COM Class 11
 - in EPI COM Class 14
- ESI
 - constants 170
 - data structures 170
 - functions 172
- Event parameter
 - CICS_EpiGetEvent function 165
- except (parameter)
 - in handleException 62, 65
- exCode
 - in CclECI class 62
 - in CclEPI class 64
 - in CclException class 67
 - in CclMap class 78
 - in CclTerminal class 91
 - in Public methods 62, 64, 67, 78, 91
- ExCode 11, 14
 - in EPI COM Class 13
 - in ErrorWindow 11, 14
 - in Methods
 - in ECI COM Class 12
 - in EPI COM Class 15
 - in Map COM Class 26
 - in Terminal COM Class 38
 - in State 16, 35

- exCodeText
 - in CclECI class 62
 - in CclEPI class 64
 - in CclException class 67
 - in CclMap class 79
 - in CclTerminal class 91
 - in Public methods 62, 64, 67, 79, 91

- ExCodeText
 - in ExCode 12
 - in Methods
 - in ECI COM Class 12
 - in EPI COM Class 15
 - in Terminal COM Class 38

- EXEC CICS CONVERSE 167
- EXEC CICS RECEIVE 160, 162, 167
- EXEC CICS RECEIVE BUFFER 167
- EXEC CICS SEND 167

- exObject
 - in CclException class 67
 - in Public methods 67

- ExpiryTime
 - in SecAttr COM class 31

- extensible
 - in CclBuf 50, 51
 - in CclBuf class 50
 - in dataAreaType 52
 - in DataAreaType 56
 - in setDataLength 55

- external
 - in dataAreaOwner 52
 - in DataAreaOwner 55

- ExtractString
 - in Methods
 - in Buffer COM Class 2

F

- failed
 - in EndTerminalReason 96

- false
 - in ErrorWindow 11, 14

- False
 - in Poll 24, 39

- FALSE
 - in Validate 27

- field
 - in CclMap class 79
 - in CclScreen class 81
 - in Public methods 79, 81

- Field
 - in Field COM Class 17
 - in Screen COM Class 27

- Field COM class
 - Methods
 - AppendText 17
 - BackgroundColor 18

- Field COM class (*continued*)

- Methods (*continued*)
 - BaseAttribute 18
 - Column 18
 - DataTag 18
 - ForegroundColor 19
 - Highlight 19
 - InputProt 19
 - InputType 20
 - Intensity 20
 - Length 20
 - Position 20
 - ResetDataTag 21
 - Row 21
 - SetBaseAttribute 21
 - SetExtAttribute 21
 - SetText 21
 - Text 22
 - TextLength 22
 - Transparency 22

- FieldByIndex
 - in Methods
 - in Screen COM Class 28

- FieldByName
 - in Methods
 - in Map COM Class 27
 - in Validate 27

- FieldByPosition
 - in Methods
 - in Screen COM Class 29

- fieldCount
 - in CclScreen class 81
 - in Public methods 81

- FieldCount
 - in Methods
 - in Screen COM Class 29

- fields
 - in validate 80

- fields (parameter)
 - in validate 80

- fixed
 - in CclBuf 50, 51
 - in CclBuf class 50
 - in dataAreaType 52
 - in DataAreaType 56

- Flow
 - in Cancel 5
 - in Changed 5
 - in Commit 46
 - in Connect COM Class 4
 - in Flow COM class 22
 - in Flow COM Class 22
 - in Link 7
 - in SetSyncType 24, 25
 - in Status 9

- flow (parameter)
 - in backout 97
 - in BackOut 45
 - in cancel 57
 - in Cancel 5
 - in changed 57
 - in Changed 5
 - in commit 97
 - in Commit 46
 - in link 58
 - in Link 7
 - in status 60
 - in Status 9
- Flow COM class
 - Methods
 - AbendCode 23
 - CallType 23
 - CallTypeText 23
 - Diagnose 23
 - Flowid 24
 - ForceReset 24
 - Poll 24
 - SetSyncType 24
 - SetTimeout 25
 - SyncType 25
 - Timeout 25
 - Wait 25
- flowId
 - in CclFlow class 75
 - in Public methods 75
- Flowid
 - in Methods
 - in Flow COM Class 24
- forceReset
 - in CclFlow class 75
 - in CclUOW class 97
 - in Public methods 75, 97
- ForceReset
 - in Methods
 - in Flow COM Class 24
 - in UOW COM Class 46
- foregroundColor
 - in CclField class 69
 - in Public methods 69
- ForegroundColor
 - in Methods
 - in Field COM Class 19
- format (parameter)
 - in SetErrorFormat 13, 16

G

- Gateway initialization fileClient initialization file
 - in Connect 36
 - in Details 6
 - in ServerCount 12, 15

- Gateway initialization fileClient initialization file
 - (continued)
 - in ServerDesc 16
 - in ServerName 8, 16, 41
- GetDate
 - in SecTime COM Class 32
- glossary of terms and abbreviations 247
- gray
 - in Color 73
- green
 - in Color 73

H

- handleException
 - in CclECI class 62
 - in CclEPI class 65
 - in Public methods 62, 65
- handleReply
 - in CclFlow class 76
 - in CclSession class 86
 - in Public methods 76, 86
- highlight
 - in CclField class 69
 - in Public methods 69
- Highlight
 - in CclField class 73
 - in Enumerations 73
 - in Methods
 - in Field COM Class 19
- Hours
 - in SecTime COM Class 32
- Hundredths
 - in SecTime COM Class 32

I

- idle
 - in send 94
 - in State 87, 96
- in Buffer COM Class 1
- inactive
 - in CallType 77
- index (parameter)
 - in ExCode 15
 - in field 79, 81
 - in FieldByIndex 28
 - in namedField 79
 - in serverDesc 63, 65
 - in ServerDesc 12, 13, 15, 16
 - in serverName 63, 65
 - in ServerName 13, 16
 - in validate 80
- initEPI
 - in CclEPI constructor 64
 - in CclTerminal constructor 88

- initialization file
 - in CclTerminal constructor 88
 - in serverCount 63, 65
 - in serverDesc 65
 - in serverName 65
- inputProt
 - in CclField class 70
 - in Public methods 70
- InputProt
 - in Methods
 - in Field COM Class 19
- inputType
 - in CclField class 70
 - in Public methods 70
- InputType
 - in Methods
 - in Field COM Class 20
- insert
 - in CclBuf class 53
 - in Public methods 53
- InsertString
 - in Methods
 - in Buffer COM Class 2
- install
 - in CclTerminal class 91
 - in Public methods 91
- Install
 - in Methods
 - in Terminal COM Class 38
- install_path vii
- installation
 - default location vii
 - path vii
- installation path vii
- instance
 - in CclECI class 63
 - in Public methods 63
- intense
 - in BaseInts 72
 - in intensity 70
- intenseHlt
 - in Highlight 73
- intensity
 - in CclField class 70
 - in Public methods 70
- Intensity
 - in Methods
 - in Field COM Class 20
- internal
 - in CclBuf 51
 - in dataAreaOwner 52
 - in DataAreaOwner 55
- invalidMap
 - in CclMap constructor 78

- invalidState
 - in poll 92
 - in send 94

J

- Javadoc 47, 181

K

- key (parameter)
 - in setAID 82
 - in SetAID 29, 30

L

- labels
 - in validate 80
- LastVerifiedTime
 - in SecAttr COM class 31
- len
 - in validate 80
- length
 - in CclField class 70
 - in Public methods 70
- Length
 - in Methods
 - in Buffer COM Class 3
 - in Field COM Class 20
- length (parameter)
 - in appendText 68
 - in assign 51
 - in CclBuf 50, 51
 - in cut 52
 - in ExtractString 2
 - in insert 53
 - in replace 55
 - in setDataLength 55
 - in SetLength 3
 - in setText 71
- link
 - in CallType 78
 - in CclConn class 58
 - in Public methods 58
 - in SetSyncType 25
- Link
 - in Details 6
 - in Methods
 - in Connect COM Class 6
 - in Poll 24
 - in UOW COM class 45
- List parameter 133
 - CICS_EciListSystems 132
 - CICS_EpiListSystems function 146
- listState
 - in CclBuf class 53
 - in CclConn class 59
 - in CclECI class 63
 - in CclFlow class 76

listState (*continued*)
in CclUOW class 98
in Public methods 53, 59, 63, 76, 98

M

makeSecurityDefault
in CclConn class 59
in CclTerminal class 91
in Public methods 59, 91

MakeSecurityDefault
in Methods
in Connect COM Class 7
in Terminal COM Class 39

Map
in Map COM Class 25

map (parameter)
in validate 80

Map COM class
Methods
ExCode 26
FieldByName 27
Validate 27

mapName
in CclScreen class 82
in Public methods 82

MapName
in Methods
in Screen COM Class 29

mapname (parameter)
in Validate 27

mapSetName
in CclScreen class 82
in Public methods 82

MapSetName
in Methods
in Screen COM Class 29

MaxBufferSize (parameter)
in CclBuf class 50

maxRequests
in CclTerminal constructor 88

maxServers
in serverDesc 65
in serverName 65

methodName
in CclException class 68
in Public methods 68

Methods
AbendCode 23
AlterSecurity 5, 35
AppendString 2
AppendText 17
BackgroundColor 18
BackOut 45
BaseAttribute 18
CallType 23

Methods (*continued*)
CallTypeText 23
Cancel 5
CCSID 36
Changed 5
ChangePassword 5, 36
Column 18
Commit 46
Connect 36
CursorCol 28
CursorRow 28
Data 2
DataTag 18
Depth 28
Details 6
Devtype 37
Diagnose 14, 23, 34, 37
Disconnect 37
DisconnectWithPurge 37
DiscReason 37
ErrorFormat 11, 14
ErrorOffset 11, 14
ErrorWindow 11, 14
ExCode 12, 15, 26, 38
ExCodeText 12, 15, 38
ExtractString 2
FieldByIndex 28
FieldByName 27
FieldByPosition 29
FieldCount 29
Flowid 24
ForceReset 24, 46
ForegroundColor 19
Highlight 19
in Buffer COM Class 2
in Connect COM Class 5
in ECI COM Class 11
in EPI COM Class 14
in Field COM Class 17
in Flow COM Class 23
in Map COM Class 26
in Screen COM Class 28
in Session COM Class 34
in Terminal COM Class 35
in UOW COM Class 45
InputProt 19
InputType 20
InsertString 2
Install 38
Intensity 20
Length 3, 20
Link 6
MakeSecurityDefault 7, 39
MapName 29
MapSetName 29

Methods (continued)

- NetName 39
- Overlay 3
- Password 7, 39
- Poll 24, 39
- PollForReply 40
- Position 20
- QueryATI 40
- ReadTimeout 40
- ReceiveATI 40
- ResetDataTag 21
- Row 21
- Screen 41
- Send 41
- ServerCount 12, 15
- ServerDesc 12, 15
- ServerName 8, 13, 16, 41
- ServerStatus 8
- ServerStatusText 8
- SetAID 29
- SetATI 41
- SetBaseAttribute 21
- SetCursor 30
- SetData 3
- SetErrorFormat 13, 16
- SetExtAttribute 21
- SetLength 3
- SetString 4
- SetSyncType 24, 34
- SetTermDefns 42
- SetText 21
- SetTimeout 25
- SignonCapability 43
- Start 43
- State 16, 34, 44
- Status 8
- String 4
- SyncType 25
- TermId 44
- Terminate 17
- Text 22
- TextLength 22
- Timeout 25
- TranDetails 9
- TransId 35, 44
- Transparency 22
- UnpaddedPassword 9
- UnpaddedServerName 9
- UnpaddedUserid 10
- UowId 46
- Userid 44
- UserId 10
- Validate 27
- VerifyPassword 10, 44
- Wait 25

Methods (continued)

- Width 30
- migration vii
- Minutes
 - in SecTime COM Class 33
- modified
 - in BaseMDT 72
 - in dataTag 69
- Month
 - in SecTime COM Class 33
- multipleInstance
 - in CclECI class 61

N

- n (parameter)
 - in position 70
- name (parameter)
 - in FieldByName 27
- namedField
 - in CclMap class 79
 - in Protected methods 79
- NameSpace parameter
 - CICS_EciListSystems 132
 - CICS_EpiAddTerminal function 147
 - CICS_EpiListSystems function 145
- netName
 - in CclTerminal class 92
 - in Public methods 92
- NetName
 - in Methods
 - in Terminal COM Class 39
- netname (parameter)
 - in CclTerminal constructor 87, 88
- NetName parameter
 - CICS_EpiAddExTerminal function 151
 - CICS_EpiAddTerminal function 147
- neutral
 - in Color 73
- New
 - in Buffer COM Class 1
- newPassword (parameter)
 - in alterSecurity 57
 - in AlterSecurity 5, 35
 - in changed 58
 - in ChangePassword 6
 - in changePassword method 89
- NewPassword parameter
 - CICS_ChangePassword function 175
- newstate (parameter)
 - in setATI 94
- newUserid (parameter)
 - in alterSecurity 57
 - in AlterSecurity 5, 35
- no
 - := 54

- no (*continued*)
 - in Bool 49
 - in poll 76
- normal
 - in BaseInts 72
 - in intensity 70
- normalHlt
 - in Highlight 73
- notDiscon
 - in EndTerminalReason 96
- Nothing
 - in Poll 24
- NotifyFn parameter
 - CICS_EpiAddExTerminal function 152
 - CICS_EpiAddTerminal function 149
- numeric
 - in BaseType 73
 - in inputType 70
- nworkName (parameter)
 - in Connect 36, 37
 - in SetTermDefns 42

O

- off
 - in Bool 49
- offset (parameter)
 - in cut 52
 - in dataArea 52
 - in ExtractString 2
 - in insert 53
 - in InsertString 2
 - in Overlay 3
 - in replace 55
- OldPassword parameter
 - CICS_ChangePassword function 175
- on
 - in Bool 49
- opaqueTran
 - in Transparency 73
- orange
 - in Color 73
- orTran
 - in Transparency 73
- outofService
 - in EndTerminalReason 96
- Overlay
 - in Methods
 - in Buffer COM Class 3

P

- PA1
 - in AID 83
- PA3
 - in AID 83

- paleCyan
 - in Color 73
- paleGreen
 - in Color 73
- parameter
 - in CclMap constructor 78
 - in CclTerminal constructor 88
 - in send 93
 - in setCursr 82
 - in setExtAttribute 71
- password 58
 - in CclConn class 59
 - in CclTerminal class 92
 - in Public methods 59, 92
 - in verifyPassword method 61
- Password
 - in Methods
 - in Connect COM Class 7
 - in Terminal COM Class 39
- password (parameter)
 - in alterSecurity method 89
 - in CclConn constructor 56, 57
 - in CclTerminal constructor 88
 - in ChangePassword 36
 - in Details 6
 - in SetTermDefns 42
- Password parameter
 - CICS_EpiSetSecurity function 158
 - CICS_SetDefaultSecurity function 178
 - CICS_VerifyPassword function 172
- PF1
 - in AID 83
- PF24
 - in AID 83
- pink
 - in Color 73
- poll
 - in CclFlow class 76
 - in CclTerminal class 92
 - in Public methods 76, 92
- Poll
 - in Methods
 - in Flow COM Class 24
 - in Terminal COM Class 39
 - in SetSyncType 25, 34
- PollForReply
 - in Methods
 - in Terminal COM Class 40
- position
 - in CclField class 70
 - in Public methods 70
- Position
 - in Methods
 - in Field COM Class 20

- programming
 - reference 47, 181
- programName (parameter)
 - in link 58
 - in Link 7
- protect
 - in BaseProt 73
 - in inputProt 70
- Protected methods
 - in CclMap class 79
 - namedField 79
 - validate 80
- Public methods
 - := 54
 - abendCode 67, 75
 - alterSecurity 57, 89
 - appendText 68
 - assign 51
 - backgroundColor 69
 - backout 97
 - baseAttribute 69
 - callType 75
 - callTypeText 75
 - cancel 57
 - CCSid 90
 - change password 58
 - changed 57
 - changePassword 89
 - className 67
 - column 69
 - commit 97
 - connection 75
 - cursorCol 81
 - cursorRow 81
 - cut 52
 - dataArea 52
 - dataAreaLength 52
 - dataAreaOwner 52
 - dataAreaType 52
 - dataLength 53
 - dataTag 69
 - depth 81
 - diagnose 64, 67, 75, 86, 90
 - disconnect 90
 - discReason 90
 - exCode 62, 64, 67, 78, 91
 - exCodeText 62, 64, 67, 79, 91
 - exObject 67
 - field 79, 81
 - fieldCount 81
 - flowId 75
 - forceReset 75, 97
 - foregroundColor 69
 - handleException 62, 65
 - handleReply 76, 86
 - Public methods (*continued*)
 - highlight 69
 - in CclBuf class 51
 - in CclConn class 57
 - in CclECI class 62
 - in CclEPI class 64
 - in CclException class 67
 - in CclField class 68
 - in CclFlow class 75
 - in CclMap class 78
 - in CclScreen class 81
 - in CclSession class 86
 - in CclTerminal class 89
 - in CclUOW class 97
 - inputProt 70
 - inputType 70
 - insert 53
 - install 91
 - instance 63
 - intensity 70
 - length 70
 - link 58
 - listState 53, 59, 63, 76, 98
 - makeSecurityDefault 59, 91
 - mapName 82
 - mapSetName 82
 - methodName 68
 - netName 92
 - password 59, 92
 - poll 76, 92
 - position 70
 - queryATI 92
 - r= 53
 - readTimeout 93
 - receiveATI 93
 - replace 55
 - resetDataTag 71
 - row 71
 - screen 93
 - send 93, 94
 - serverCount 63, 65
 - serverDesc 63, 65
 - serverName 59, 60, 63, 65, 94
 - serverStatus 60
 - serverStatusText 60
 - setAID 82
 - setATI 94
 - setBaseAttribute 71
 - setCursor 82
 - setDataLength 55
 - setExtAttribute 71
 - setText 71, 72
 - setTimeout 77
 - signonCapability 94
 - state 66, 86, 94

Public methods (*continued*)

- status 60
- syncType 77
- termID 95
- terminal 86
- terminate 66
- text 72
- textLength 72
- timeout 77
- transID 86, 95
- transparency 72
- uow 77
- uowId 98
- userId 60, 61, 95
- verifyPassword 61, 95
- wait 77
- width 83
- publications 239
- purple
 - in Color 73

Q

- queryATI
 - in CclTerminal class 92
 - in Public methods 92
- QueryATI
 - in Methods
 - in Terminal COM Class 40

R

- r=
 - in CclBuf class 53
 - in Public methods 53
- readTimeout
 - in CclTerminal class 93
 - in Public methods 93
- ReadTimeout
 - in Methods
 - in Terminal COM Class 40
- readTimeOut (parameter)
 - in CclTerminal constructor 88
- ReadTimeout (parameter)
 - in SetTermDefns 42
- receiveATI
 - in CclTerminal class 93
 - in Public methods 93
- ReceiveATI
 - in Methods
 - in Terminal COM Class 40
- red
 - in Color 73
- replace
 - in CclBuf class 55
 - in Public methods 55

- reserved1
 - field in ECI parameter block 100
 - with ECL_ASYNC call type 111
 - with ECL_STATE_ASYNC call type 120
 - with ECL_STATE_SYNC call type 116
 - with ECL_SYNC call type 104
- resetDataTag
 - in CclField class 71
 - in Public methods 71
- ResetDataTag
 - in Methods
 - in Field COM Class 21
 - in ResetDataTag 21
- reverseHlt
 - in Highlight 73
- row
 - in CclField class 71
 - in Public methods 71
 - in validate 80
- Row
 - in Methods
 - in Field COM Class 21
- row (parameter)
 - in field 79, 81
 - in setCursor 82
- rowPos (parameter)
 - in FieldByPosition 29
 - in SetCursor 30
- runTran (parameter)
 - in CclConn constructor 56, 57
 - in TranDetails 9

S

- screen
 - in CclTerminal class 93
 - in Public methods 93
- Screen
 - in Field COM Class 17
 - in Methods
 - in Terminal COM Class 41
 - in Screen 41
 - in Screen COM class 27, 28
 - in Validate 27
- screen (parameter)
 - in CclMap constructor 78
 - in handleReply 86
- Screen COM class
 - Methods
 - CursorCol 28
 - CursorRow 28
 - Depth 28
 - FieldByIndex 28
 - FieldByPosition 29
 - FieldCount 29
 - MapName 29

Screen COM class (*continued*)

- Methods (*continued*)
 - MapSetName 29
 - SetAID 29
 - SetCursor 30
 - Width 30
- Screen.fieldbyIndex method
 - in Ccl Field COM class 17
- screenRef (parameter)
 - in Validate 27
- Seconds
 - in SecTime COM Class 33
- send
 - in CclTerminal class 93, 94
 - in Public methods 93, 94
- Send
 - in Methods
 - in Terminal COM Class 41
 - in Poll 39
 - in SetSyncType 34
- server
 - in poll 92
 - in State 87, 96
- server (parameter)
 - in CclTerminal constructor 87, 88
- serverCount
 - in CclECI class 63
 - in CclEPI class 65
 - in Public methods 63, 65
- ServerCount
 - in Methods
 - in ECI COM Class 12
 - in EPI COM Class 15
- serverDesc
 - in CclECI class 63
 - in CclEPI class 65
 - in Public methods 63, 65
- ServerDesc
 - in ExCode 15
 - in Methods
 - in ECI COM Class 12
 - in EPI COM Class 15
- serverName
 - in CclConn class 59, 60
 - in CclECI class 63
 - in CclEPI class 65
 - in CclTerminal class 94
 - in Public methods 59, 60, 63, 65, 94
- ServerName
 - in Details 6
 - in ExCode 15
 - in Methods
 - in Connect COM Class 8
 - in ECI COM Class 13
 - in EPI COM Class 16
- ServerName (*continued*)
 - in Methods (*continued*)
 - in Terminal COM Class 41
- serverName (parameter)
 - in CclConn constructor 56
 - in Details 6
- serverStatus
 - in CclConn class 60
 - in Public methods 60
- ServerStatus
 - in CclConn class 61
 - in Enumerations 61
 - in Methods
 - in Connect COM Class 8
- serverStatusText
 - in CclConn class 60
 - in Public methods 60
- ServerStatusText
 - in Methods
 - in Connect COM Class 8
- servName (parameter)
 - in Connect 36
 - in SetTermDefns 42
- Session
 - in Send 41
 - in Session COM Class 33
 - in SetSyncType 34
 - in Start 43
 - in State 44
- session (parameter)
 - in ReceiveATI 40
 - in receiveATI method 93
 - in send 93, 94
 - in Send 41
 - in Start 43
- Session COM class
 - Methods
 - Diagnose 34
 - SetSyncType 34
 - State 34
 - TransId 35
- setAID
 - in CclScreen class 82
 - in Public methods 82
- SetAID
 - in Methods
 - in Screen COM Class 29
- setATI
 - in CclTerminal class 94
 - in Public methods 94
- SetATI
 - in Methods
 - in Terminal COM Class 41
- setBaseAttribute
 - in CclField class 71

- setBaseAttribute (*continued*)
 - in Public methods 71
- SetBaseAttribute
 - in Methods
 - in Field COM Class 21
- setCursor
 - in CclScreen class 82
 - in Public methods 82
- SetCursor
 - in Methods
 - in Screen COM Class 30
- SetData
 - in Methods
 - in Buffer COM Class 3
- setDataLength
 - in CclBuf class 55
 - in Public methods 55
- SetErrorFormat
 - in Methods
 - in ECI COM Class 13
 - in EPI COM Class 16
- setExtAttribute
 - in CclField class 71
 - in Public methods 71
- SetExtAttribute
 - in Methods
 - in Field COM Class 21
- SetLength
 - in Buffer COM Class 1
 - in Methods
 - in Buffer COM Class 3
- SetString
 - in Methods
 - in Buffer COM Class 4
- SetSyncType
 - in Methods
 - in Flow COM Class 24
 - in Session COM Class 34
- SetTermDefns
 - in Methods
 - in Terminal COM Class 42
- setText
 - in CclField class 71, 72
 - in Public methods 71, 72
- SetText
 - in Methods
 - in Field COM Class 21
- setTimeout
 - in CclFlow class 77
 - in Public methods 77
- SetTimeout
 - in Methods
 - in Flow COM Class 25
- shutdown
 - in EndTerminalReason 96
- signoff
 - in EndTerminalReason 96
- signonCapability
 - in CclTerminal class 94
 - in Public methods 94
- SignonCapability
 - in Methods
 - in Terminal COM Class 43
- signonCapability (parameter)
 - in CclTerminal constructor 88
 - in SetTermDefns 42
- signonCapable
 - in signonType 96
- signonIncapable
 - in signonType 96
- signonType
 - in CclTerminal class 95
 - in Enumerations 95
- signonUnknown
 - in signonType 96
- Size parameter
 - CICS_EpiReply function 162
 - CICS_EpiStartTran function 161
- stackPages (parameter)
 - in CclFlow 74
- Start
 - in Methods
 - in Terminal COM Class 43
 - in Poll 39
 - in SetSyncType 34
- startdata (parameter)
 - in send 93
- startData (parameter)
 - in Start 43
- state
 - in CclEPI class 66
 - in CclSession class 86
 - in CclTerminal class 94
 - in Public methods 66, 86, 94
- State
 - in CclEPI class 66
 - in CclSession class 87
 - in CclTerminal class 96
 - in Enumerations 66, 87, 96
 - in EPI COM Class 13
 - in Methods
 - in EPI COM Class 16
 - in Session COM Class 34
 - in Terminal COM Class 44
 - in State 44
- state (parameter)
 - in handleReply 86
- stateVal (parameter)
 - in SetATI 41

- status
 - in CallType 78
 - in CclConn class 60
 - in Public methods 60
 - in ServerStatus 8
 - in ServerStatusText 8
 - in SetSyncType 25
- Status
 - in Details 6
 - in Methods
 - in Connect COM Class 8
 - in Poll 24
- String
 - in Methods
 - in Buffer COM Class 4
- string (parameter)
 - in AppendString 2
 - in InsertString 2
 - in Overlay 3
 - in SetString 4
- sync
 - in CclSession constructor 86
 - in Sync 49
- Sync
 - in Ccl class 49
 - in Enumerations 49
- syncType
 - in CclFlow class 77
 - in poll 76, 92
 - in Public methods 77
 - in wait 77
- SyncType
 - in Methods
 - in Flow COM Class 25
- syncType (parameter)
 - in CclFlow 74
 - in CclSession constructor 85
 - in SetSyncType 24, 34
 - in SyncType 25
- system information structure 132, 145
- System parameter
 - CICS_ChangePassword function 175
 - CICS_EpiAddExTerminal function 151
 - CICS_EpiAddTerminal function 147
 - CICS_SetDefaultSecurity function 178
 - CICS_VerifyPassword function 173
- SystemName parameter
 - CICS_EciListSystems 132
- Systems parameter 133
 - CICS_EciListSystems 132
 - CICS_EpiListSystems function 145

T

- termDefined
 - in State 96

- termID
 - in CclTerminal class 95
 - in Public methods 95
- TermId
 - in Methods
 - in Terminal COM Class 44
- terminal
 - in CclSession class 86
 - in Public methods 86
- Terminal
 - in EPI COM Class 13
 - in Screen COM Class 28
 - in ServerName 41
- Terminal COM class
 - Methods
 - AlterSecurity 35
 - CCSID 36
 - ChangePassword 36
 - Connect 36
 - Devtype 37
 - Diagnose 37
 - Disconnect 37
 - DisconnectWithPurge 37
 - DiscReason 37
 - ExCode 38
 - ExCodeText 38
 - Install 38
 - MakeSecurityDefault 39
 - NetName 39
 - Password 39
 - Poll 39
 - PollForReply 40
 - QueryATI 40
 - ReadTimeout 40
 - ReceiveATI 40
 - Screen 41
 - Send 41
 - ServerName 41
 - SetATI 41
 - SetTermDefns 42
 - SignonCapability 43
 - Start 43
 - State 44
 - TermId 44
 - TransId 44
 - Userid 44
 - VerifyPassword 44
- terminal index 147, 150
- Terminal.Connect
 - in Screen COM Class 28
- Terminal.Screen
 - in Screen COM Class 28
- terminate
 - in CclEPI class 66
 - in Public methods 66

- Terminate
 - in Methods
 - in EPI COM Class 17
 - in Terminate 17
- TermIndex parameter
 - CICS_EpiAddExTerminal function 153
 - CICS_EpiAddTerminal function 149
 - CICS_EpiATISState function 164
 - CICS_EpiDelTerminal function 156
 - CICS_EpiGetEvent function 165
 - CICS_EpiInquireSystem function 155
 - CICS_EpiPurgeTerminal function 157
 - CICS_EpiReply function 162
 - CICS_EpiSetSecurity function 158
 - CICS_EpiStartTran function 160
- text
 - in CclField class 72
 - in Public methods 72
- Text
 - in Methods
 - in Field COM Class 22
- text (parameter)
 - := 54
 - in appendText 68
 - in CclBuf 51
 - in setText 71, 72
 - r= 53, 54
- textLength
 - in CclField class 72
 - in Public methods 72
- TextLength
 - in Methods
 - in Field COM Class 22
- textString (parameter)
 - in AppendText 17, 18
 - in SetText 21, 22
- timeout
 - in CclFlow class 77
 - in Public methods 77
- Timeout
 - in Methods
 - in Flow COM Class 25
- timeout (parameter)
 - in CclFlow 74
 - in Install 38
 - in setTimeout 77
- tranCode (parameter)
 - in Start 43
- TranDetails
 - in Connect COM Class 4
 - in Methods
 - in Connect COM Class 9
- transID
 - in CclSession class 86
 - in CclTerminal class 95
- transID (*continued*)
 - in Public methods 86, 95
- TransId
 - in Methods
 - in Session COM Class 35
 - in Terminal COM Class 44
- transid (parameter)
 - in send 93
- TransId parameter
 - CICS_EpiStartTran function 160
- transparency
 - in CclField class 72
 - in Public methods 72
- Transparency
 - in CclField class 73
 - in Enumerations 73
 - in Methods
 - in Field COM Class 22
- true
 - in ErrorWindow 11, 14
- True
 - in Poll 24, 39
- TRUE
 - in Validate 27
- txnTimedOut
 - in State 96
- type (parameter)
 - in CclBuf 50, 51

U

- unavailable
 - in ServerStatus 61
- underscoreHlt
 - in Highlight 73
- unit (parameter)
 - in link 58
- unitOfWork (parameter)
 - in Link 7
- unknown
 - in EndTerminalReason 96
 - in ServerStatus 61
- unknownServer
 - in CclTerminal constructor 88
- unmodified
 - in BaseMDT 72
 - in dataTag 69
- unmodified (parameter)
 - in resetDataTag 71
- unpadded (parameter)
 - in status 59, 60, 61
- UnpaddedPassword
 - in Methods
 - in Connect COM Class 9

- UnpaddedServerName
 - in Methods
 - in Connect COM Class 9
- UnpaddedUserId
 - in Methods
 - in Connect COM Class 10
- unprotect
 - in BaseProt 73
 - in inputProt 70
- uow
 - in CclFlow class 77
 - in Public methods 77
- UOW
 - in Link 7
 - in UOW COM class 45
- UOW COM class
 - Methods
 - BackOut 45
 - Commit 46
 - ForceReset 46
 - UowId 46
- uowId
 - in CclUOW class 98
 - in Public methods 98
- UowId
 - in Methods
 - in UOW COM Class 46
- user-defined
 - CICS_EpiSystemIdExit 209
 - CICS_EpiTerminateExit 197
- user-defined return code
 - CICS_EciDataReturnExit 192
 - CICS_EciDataSendExit 191
 - CICS_EciExternalCallExit1 187
 - CICS_EciExternalCallExit2 188
 - CICS_EciInitializeExit 184
 - CICS_EciSetProgramAliasExit 193
 - CICS_EciSystemIdExit 190
 - CICS_EciTerminateExit 185
 - CICS_EpiAddTerminalExit 199
 - CICS_EpiDelTerminalExit 206
 - CICS_EpiGetEventExit 207
 - CICS_EpiInitializeExit 196
 - CICS_EpiReplyExit 205
 - CICS_EpiStartTranExit 204
 - CICS_EpiStartTranExtendedExit 203
 - CICS_EpiTermIdExit 201
 - CICS_EpiTermIdInfoExit 202
 - CICS_EpiTranFailedExit 211
- userId
 - in CclConn class 60, 61
 - in CclTerminal class 95
 - in Public methods 60, 61, 95
- Userid
 - in Methods
 - in Terminal COM Class 44
- UserId
 - in Methods
 - in Connect COM Class 10
- userid (parameter)
 - in alterSecurity method 89
 - in CclTerminal constructor 88
 - in SetTermDefns 42
- userId (parameter)
 - in CclConn constructor 56
 - in Details 6
- userID (parameter)
 - in CclConn constructor 57
 - in Details 6
- UserId parameter
 - CICS_ChangePassword function 175
 - CICS_EpiSetSecurity function 158
 - CICS_SetDefaultSecurity function 178
 - CICS_VerifyPassword function 172

V

- validate
 - in CclMap class 80
 - in Protected methods 80
- Validate
 - in Methods
 - in Map COM Class 27
- value (parameter)
 - in setExtAttribute 71
- Value (parameter)
 - in SetExtAttribute 21
- verifyPassword
 - in CclConn class 61
 - in CclTerminal class 95
 - in Public methods 61, 95
- VerifyPassword
 - in Methods
 - in Connect COM Class 10
 - in Terminal COM Class 44
- Version parameter
 - CICS_EpiInitialize function 144

W

- wait
 - in CclFlow class 77
 - in Public methods 77
- Wait
 - in Methods
 - in Flow COM Class 25
 - in Wait 25
- Wait parameter
 - CICS_EpiGetEvent function 165

- white
 - in Color 73
- width
 - in CclScreen class 83
 - in Public methods 83
 - in validate 80
- Width
 - in Methods
 - in Screen COM Class 30
- withPurge
 - in disconnect method
 - in CclTerminal class 90
- WorkLoad Manager 192

X

- xorTran
 - in Transparency 73

Y

- Year
 - in SecTime COM Class 33
- yellow
 - in Color 73
- yes
 - ;= 54
 - in Bool 49
 - in poll 76

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

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

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 information 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.

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

COPYRIGHT LICENSE:

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

Trademarks

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

AIX
AnyNet
AS/400
CICS
CICS/400
CICS/ESA
CICS/VSE
DB2
Domino
Hummingbird
IBM
IBM
IBMLink
IMS
iSeries
MQSeries
MVS
MVS/ESA
Notes
OS/2
OS/390
POWER
pSeries
RACF
Redbooks
RETAIN
RMF
RS/6000
SAA
SP2
System/390
Tivoli
TXSeries
VisualAge
VSE/ESA
VTAM
WebSphere
z/OS
zSeries

Microsoft, Windows, Windows NT[®], and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Intel[®], Intel Inside[®] (logos), MMX and Pentium[®] are trademarks of Intel Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Sending your comments to IBM

If you especially like or dislike anything about this book, use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Limit your comments to the information in this book and the way in which the information is presented.

To ask questions, make comments about the functions of IBM products or systems, or to request additional publications, contact your IBM representative or your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:
User Technologies Department (MP095)
IBM United Kingdom Laboratories
Hursley Park
WINCHESTER,
Hampshire
SO21 2JN
United Kingdom
- By fax:
 - +44 1962 842327 (if you are outside the UK)
 - 01962 842327 (if you are in the UK)
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
 - IBMLink™: HURSLEY(IDRCF)
 - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.



Program Number: 5724-I81, 5655-R25 and 5724-J09

SC34-6759-02



Spine information:



CICS Transaction Gateway

Programming Reference

Version 7.1