CICS Transaction Gateway for Multiplatforms
Version 9 Release 2

Programming Reference

SC34-7340-00

CICS Transaction Gateway for Multiplatforms
Version 9 Release 2

Programming Reference

SC34-7340-00

Note

Note: Before you use this information and the product it supports, read the information in Safety and environmental notices and Notices.

This edition applies to Version 9, Release 2 Modification 0 of CICS TG for Multiplatforms, program number 5724-I81 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this information

This information describes the planning, installation, configuration, and operation of the IBM® CICS® Transaction Gateway and the IBM CICS Transaction Gateway Desktop Edition products.

You should be familiar with the operating system on which CICS Transaction Gateway runs and also with Internet terminology.

This information includes trademarks including Java™, for more information about Trademarks, see the Trademark information at the back of this publication.

Part 1. Programming Reference

Chapter 1. C

Programming reference information for the CICS Transaction Gateway C APIs.

ECI V1

This section describes CICS_ExternalCall and the call types that control its function.

CICS_ExternalCall (ECI_Parms)

`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_abend_code
Abend code for a failed program.

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

eci_call_type
An integer field defining the type of call being made.

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_extend_mode
Used to manage logical units of work that span multiple ECI requests. See the information about managing logical units of work in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* for more details.

eci_luw_token
An identifier for a logical unit of work.

eci_message_qualifier
A user-provided reference to an asynchronous call.

eci_password
Password for security checking.

eci_password2
Password for security checking. This is used if the password is more than 8 characters.

eci_program_name
The name of a program to be called.

eci_sysid
Reserved for future use; leave null.

eci_system_name
The name of a 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 information about timeout of the ECI request in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*. For remote mode IPIC, this value can be overridden by the ECITIMEOUT property on the IPIC server definition.

eci_tpn
A transaction identifier for the mirror transaction.

eci_transid
A transaction identifier.

eci_userid
User ID for security checking.

eci_userid2
User ID for security checking. This is used if the User ID is more than 8 characters.

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

reserved1
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.

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 be in the CICS Transaction Gateway or in the server. Save the information returned in the CICS Transaction Gateway's error log, because 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 files:

C `<install_path>/include/cics_eci.h`

COBOL
`<install_path>/copybook/cicseci.cbl`

Call types

Call types define the action requested by the call.

ECI_SYNC call type

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

Set the ECI parameter block 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 information about managing logical units of work in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* 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 uppercase 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 uppercase 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 uppercase 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 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 ends abnormally. 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. Application developers are advised to use a maximum size of 32,500 bytes because this is guaranteed to be flowed successfully across all protocols. COMMAREA sizes greater than this might cause an ECI_ERR_INVALID_DATA_LENGTH return code to be generated.

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 ends abnormally.

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 information about managing logical units of work in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* 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 each case, changes to recoverable resources are committed by a CICS end-of-task sync point, 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 each 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 is used as input to all later calls related to the same logical unit of work. (See the information about managing logical units of work in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* 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 initialize this field 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 is one of the server names returned by **CICS_EciListSystems**. The value can 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 CICS 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 in the unit of work. Subsequent requests within the same unit of work will use the mirror transaction specified on 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 3.

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 ended abnormally. The abend code is 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
The maximum number of concurrent requests handled by the Client daemon, as defined by the configuration parameter **maxrequests** in the configuration file, has been reached.

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

The ECI_ASYNC call type provides an asynchronous program link call to start, continue, or end a logical unit of work.

Purpose

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. Perform the minimum possible processing within the callback routine.
2. ECI functions cannot be invoked from within the callback routine.
3. The callback routine indicates 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 information about managing logical units of work in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* 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 uppercase 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 uppercase 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 uppercase 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 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. Application developers are advised to use a maximum size of 32,500 bytes, as this is guaranteed to be flowed successfully across all protocols. COMMAREA sizes greater than this may generate an ECI_ERR_INVALID_DATA_LENGTH return code.

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 ends abnormally.

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 information about managing logical units of work in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* 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 each case, changes to recoverable resources are committed by a CICS end-of-task sync point, 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 each 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, use this as input to all later calls related to the same logical unit of work. (See the information about managing logical units of work in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* 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 initialize this field 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 might 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 CICS 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 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 in the unit of work. Subsequent requests within the same unit of work will use the mirror transaction specified on 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 3.

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

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

Set the ECI parameter block 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 the information about status information calls in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* and "ECI status block" on page 28, 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 the information about status information calls in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* and "ECI status block" on page 28, 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 the information about status information calls in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* and "ECI status block" on page 28, 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 the information about status information calls in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* and "ECI status block" on page 28, 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, initialize this field 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 is one of the server names returned by **CICS_EciListSystems**. The value might be supplied whenever **eci_luw_token** is set to zero.

If the field is set to nulls, the default CICS 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 3.

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

The ECI_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.

Purpose

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 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.

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. Perform the minimum possible processing within the callback routine.
2. ECI functions cannot be invoked from within the callback routine.
3. The callback routine 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 setting the input parameter fields.

eci_call_type
Required input parameter.

Must be set to ECI_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 the information about status information calls in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* and "ECI status block" on page 28 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 the information about status information calls in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* and "ECI status block" on page 28, 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 the information about status information calls in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* and "ECI status block" on page 28, 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 the information about status information calls in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* and "ECI status block" on page 28, 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 initialize this field 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 is one of the server names returned by **CICS_EciListSystems**. The value might be supplied whenever **eci_luw_token** is set to zero.

If the field is set to nulls, the default CICS 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 3.

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

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.)

Purpose

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

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. 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 the information about status information calls in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* and "ECI status block" on page 28, 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 must not exceed 32,500. (Some client/server combinations might 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 initialize this field 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 3.

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 resources to satisfy the asynchronous request to which this reply relates. The maximum number of concurrent requests handled by the Client daemon, as defined by the configuration parameter **maxrequests** in the configuration file, has been reached.

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 user ID and password on the asynchronous request to which this reply relates.

ECI_GET_REPLY_WAIT call type

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.)

Purpose

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

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.)

Purpose

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 the information about status information calls in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* and "ECI status block" on page 28, 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 must not exceed 32,500. (Some client/server combinations might 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 initialize this field 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 3.

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 ended abnormally. The abend code is 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 resources to satisfy the asynchronous request to which this reply relates. The maximum number of concurrent requests handled by the Client daemon, as defined by the configuration parameter **maxrequests** in the configuration file, has been reached.

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 user ID and password on the asynchronous request to which this reply relates.

ECI_GET_SPECIFIC_REPLY_WAIT call type

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**.

Purpose

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.

CICS_EciListSystems (NameSpace Systems List)

Purpose

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 spaces 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.

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.

EPI

This section describes the constants and data structures that you need to use the EPI, the functions provided by the EPI that can be called from an application program, and the EPI events that occur when CICS has data to pass to the EPI application.

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 35.

EPI constants

The following constants are referred to symbolically in the descriptions of the EPI data structures, functions, and events in this section.

The values given here are to help you understand the descriptions. However, your code uses 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 the information about EPI versions in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*.)

- 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:

The **CICS_EpiSystem_t** structure contains the name and description of a CICS server.

Purpose

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 is 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:

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 can start server-provided sign-on and sign-off 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 the information about Supported conversions in the *CICS Transaction Gateway: UNIX and Linux Administration*.

UserId
A string specifying the user ID to be associated with the terminal resource. If the user ID 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:

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 IBM 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 can 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 is not specified in the model terminal definition, a default value of 12000 is assumed.

ErrLastLine
1 if the terminal resource will display error messages on its last row, 0 otherwise.

ErrIntensify
1 if the terminal resource will display error messages intensified, 0 otherwise.

ErrColor
The 3270 attribute defining the color to be used to display error messages.

ErrHilight
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
The application might start server-provided sign-on and sign-off transactions at the terminal resource.

CICS_EPI_SIGNON_INCAPABLE
The application might not start server-provided sign-on and sign-off transactions at the terminal resource.

CICS_EPI_SIGNON_UNKNOWN
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:

The **CICS_EpiEventData_t** structure holds details of a terminal-related event.

Purpose

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 54.

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 set the **Data** parameter 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 might 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: Set the **Data** and **Size** fields 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.

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 files:

C `/include/cics_epi.h`

COBOL
`/copybook/cicsepi.cbl`

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
CICS_EpiInquireSystem	**TermIndex** **System**	ERR_BAD_INDEX ERR_FAILED 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_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_ATI_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
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_EpiATIState	**TermIndex** **ATIState**	ERR_ATI_STATE ERR_BAD_INDEX ERR_FAILED ERR_NOT_INIT ERR_IN_CALLBACK ERR_NULL_PARAM NORMAL

Table 1. Summary of EPI functions (continued)

Function name	Parameters	Return codes: CICS_EPI_
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

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 the information about EPI versions in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* 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

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 issues 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

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 31 for the contents of the structure.

EPI applications 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 must 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 must provide the storage for the array and must set the **Systems** parameter to indicate the number of elements in the array.

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.

It provides a terminal index, which is used to identify the terminal resource on all further EPI calls. It also provides the information defined in the **CICS_EpiDetails_t** data structure.

The number of terminals that you can add with this operation is limited; the maximum varies according to the resources available on the client system.

The **CICS_EpiAddTerminal** function adds terminal resources that have sign-on capability that depends on the server in which the terminal resource is installed; for example, they cannot sign on to CICS Transaction Server for z/OS® servers.

Parameters

NameSpace
A pointer reserved for future use. Ensure that it 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 CICS 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 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, 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, pad it 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, set this parameter 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 installation.

CICS_EPI_ERR_MAX_SESSIONS
The maximum number of concurrent requests handled by the Client daemon, as defined by the configuration parameter **maxrequests** in the configuration file, has been reached.

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.

It provides a terminal index, which you can use 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, can 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 CICS 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 pad it 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 set this parameter 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 set the **Details** parameter 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 user ID 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 the information about Supported conversions in the *CICS Transaction Gateway: UNIX and Linux Administration*.

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 installation.

CICS_EPI_ERR_MAX_SESSIONS
The maximum number of concurrent requests handled by the Client daemon, as defined by the configuration parameter **maxrequests** in the configuration file, has been reached.

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

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, the location of which 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

The CICS_EpiDelTerminal function deletes a previously added terminal resource.

The application does 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

The CICS_EpiPurgeTerminal function purges a previously added terminal resource.

The application does 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

The CICS_EpiSetSecurity function allows a client application to specify a user ID and password to be associated with a terminal resource previously installed as sign-on incapable.

The CICS_EpiSetSecurity function can be invoked at any time; the user ID and password will be used as further transactions are started for the terminal resource. A CICS Transaction Gateway determined user ID 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 user ID, and by implication the password, to nulls.

Note that the client application is responsible for verifying the user ID 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 user ID. If the user ID 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 user ID exceeds CICS_EPI_USERID_MAX.

CICS_EPI_NORMAL
The function completed successfully.

CICS_EpiStartTran

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, pad it 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 the information about 3270 data streams for the EPI in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*.

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 installation.

CICS_EPI_ERR_MAX_SESSIONS
The maximum number of concurrent requests handled by the Client daemon, as defined by the configuration parameter **maxrequests** in the configuration file, has been reached.

CICS_EPI_NORMAL
The function completed successfully.

CICS_EpiReply

The CICS_EpiReply function sends data from a terminal resource to a CICS transaction.

CICS_EpiReply is only 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 an abend of the conversation has occurred, but the CICS_EPI_EVENT_END_TRAN event has not yet been received by the application.

CICS_EPI_NORMAL
The function completed successfully.

CICS_EpiATIState

The CICS_EpiATIState 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 might 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 with the ATI state that is required.

The EPI uses this parameter only for input.

ATIState
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 **ATIState** 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
ATIState was a null pointer.

CICS_EPI_NORMAL
The function completed successfully.

CICS_EpiGetEvent

The CICS_EpiGetEvent function obtains information about an event that has occurred for a terminal resource.

Remember that this call can 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 happens 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 must be set to point to the data buffer that is updated with any terminal data stream associated with the event. The **Size** field must 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 the information about events and callbacks in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*. Whichever mechanism is used, the data from CICS is obtained by calling **CICS_EpiGetEvent**.

CICS_EPI_EVENT_ADD_TERM

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

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 the information about 3270 data streams for the EPI in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* for details of the data stream format.

Size The length of the data in the **Data** buffer.

CICS_EPI_EVENT_CONVERSE

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 issues 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 is 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 can be set to zero to indicate that no data was sent, but a reply is still expected.

CICS_EPI_EVENT_END_TRAN

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 typically, 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 starts 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
Typical 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

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

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.

ESI V1

This section describes the constants and data structures that you need to use the ESI, and the functions provided by the ESI that can be called from an application program.

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, you must always use the symbolic names of ESI constants provided for the programming language you are using in your code.

Lengths of fields

- CICS_ESI_PASSWORD_MAX (10)
- CICS_ESI_SYSTEM_MAX (8)
- CICS_ESI_USERID_MAX (10)

ESI data structures

The three data structures are available for use with the ESI are **CICS_EsiDate_t**, **CICS_EsiTime_t** and **CICS_EsiDetails_t**.

In the descriptions of the fields in the data structures, fields described as strings are null-terminated strings.

CICS_EsiDate_t:

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:

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:

The CICS_EsiDetails_t structure contains information returned from a successful invocation of the CICS_VerifyPassword or 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 user ID was last accessed.

LastAccessTime
The time at which the user ID was last accessed.

InvalidCount

The number of invalid password verification attempts for the user ID since the last successful password verification. This value is zero on a successful invocation of the CICS_ChangePassword function.

ESI functions

This section describes the functions provided by the ESI that can be called from an application program.

CICS_VerifyPassword

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 user ID.

CICS_VerifyPassword	
	UserId *Password* *System* *Details*

Purpose

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 which user ID to verify the password for. If the user ID 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 user ID. 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 CICS 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 user ID 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 user ID 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 user ID 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

The CICS_ChangePassword function allows a client application to change the password recorded by an external security manager for a specified user ID.

```
CICS_ChangePassword
                                        UserId
                                        OldPassword
                                        NewPassword
                                        System
                                        Details
```

Purpose

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 which user ID requires a password change. If the user ID 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 user ID. 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 user ID. 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 CICS 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 user ID 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 user ID 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 user ID 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

A client application can specify a default user ID and password to be used for ECI and EPI requests passed to the server by using the `CICS_SetDefaultSecurity` function.

```
CICS_SetDefaultSecurity
                                        UserId
                                        Password
                                        System
```

Purpose

The user ID, and the password, can be set to nulls, that is, binary zeroes. In this case the default user ID and password are unset, so that CICS Transaction Gateway acts as if no user ID and password has been set.

The user ID, 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 user ID. See the documentation for your CICS server for more information on the **`Usedfltuser`** specification.

The client application is responsible for verifying the user ID and password.

Note that the user ID and password, if required, can 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 user ID, from the CICS Transaction Gateway's pop-up window, and other similar defaults.

Parameters

UserId

A pointer to a null-terminated string that specifies the user ID to be set. If the user ID 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 user ID. 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 user ID 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 CICS 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 user ID 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 2. COBOL

COBOL headers are provided for the ECI V1, EPI and ESI V1 APIs.

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

The following table shows how the names of the calls in COBOL map to the names of the calls in C:

Interface	C	COBOL
ECI	CICS_ExternalCall	CICSEXTERNALCALL
	CICS_EciListSystems	CICSECILISTSYSTEMS
EPI	CICS_EpiInitialize	CICSEPIINITIALIZE
	CICS_EpiTerminate	CICSEPITERMINATE
	CICS_EpiListSystems	CICSEPILISTSYSTEMS
	CICS_EpiAddTerminal	CICSEPIADDTERMINAL
	CICS_EpiAddExTerminal	CICSEPIADDEXTERMINAL
	CICS_EpiInquireSystem	CICSEPIINQUIRESYSTEM
	CICS_EpiDelTerminal	CICSEPIDELTERMINAL
	CICS_EpiPurgeTerminal	CICSEPIPURGETERMINAL
	CICS_EpiSetSecurity	CICSEPISETSECURITY
	CICS_EpiStartTran	CICSEPISTARTTRAN
	CICS_EpiReply	CICSEPIREPLY
	CICS_EpiATIState	CICSEPIATISTATE
	CICS_EpiGetEvent	CICSEPIGETEVENT
ESI	CICS_VerifyPassword	CICSVERIFYPASSWORD
	CICS_ChangePassword	CICSCHANGEPASSWORD
	CICS_SetDefaultSecurity	CICSSETDEFAULTSECURITY

Chapter 3. C++

CcI 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, see "C++ Exception Objects" on page 106.

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/l 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 use 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 can be reduced by setting the *MaxBufferSize* parameter in the CICS Transaction Gateway configuration file (ctg.ini). See the information about Maximum buffer size in the *CICS Transaction Gateway: UNIX and Linux Administration* or the *CICS Transaction Gateway: Windows Administration* 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 comma1;
comma1="Some text";
comma1.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

Returns the length of the data area in bytes.

unsigned long dataAreaLength() const

dataAreaOwner

Returns an enumeration value indicating whether the data area has been allocated by the **CclBuf** constructor or has been supplied from elsewhere.

DataAreaOwner dataAreaOwner() const

Possible values are internal and external.

dataAreaType

Returns an enumeration value indicating whether the data area can be extended.

DataAreaType dataAreaType() const

Possible values are extensible and fixed.

dataLength

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

unsigned long dataLength() const

insert

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

Returns a formatted string containing the current state of the object.

const char* listState() const

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 the information about linking to a CICS server program in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*. 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. See the information about passing data

to a server program and the information about monitoring server availability in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* 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 the information about controlling server interactions in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* 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 CICS 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 user ID, if needed. The length is adjusted to 16 characters by padding with blanks or truncating, if necessary.

password
The password corresponding to the user ID 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 CPMI. 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 user ID 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 user ID

newPassword
The new password corresponding to the new user ID

Updates the user ID 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 all changed calls that were 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 user ID 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 the information about managing logical units of work in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*.

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

Returns a formatted string containing the current state of the object.

const char* listState() const

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

Informs the client that the current user ID 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.

void makeSecurityDefault()

password (1)

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

const char* password() const

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)

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

const char* serverName() const

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

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

ServerStatus serverStatus() const

serverStatusText

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

const char* serverStatusText() const

userId (1)

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

const char* userId() const

userId (2)

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

unpadded

Ccl::Yes
returns a null terminated string of the stored user ID with no space padding in the string.

Ccl::No
returns the string padded with spaces exactly as invoking the user ID 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 user ID 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.

CclECI class

One instance only of the **CclECI** class can exist. It is created by the **instance** class method. It controls the client interface to the available servers.

Subclass the **CclECI** to implement your own handleException method.

One instance only of a **CclECI** subclass can exist. Any attempt to create more than one raises a multipleInstance exception.

A CclECI object cannot be copied or assigned.

CclECI constructor (protected)

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

CclECI()

Public methods

exCode

Returns an enumeration indicating the most recent exception code.

Deprecated method

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

Ccl::ExCode exCode() const

The possible values are listed under "C++ Exception Objects" on page 106.

exCodeText

Returns a text string describing the most recent exception code.

Deprecated method

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

const char* exCodeText() const

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 must always subclass **CclECI**, and provide your own implementation of **handleException**. See the information about handling exceptions in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*. The default implementation merely throws the exception object.

instance

A class method that returns a pointer to the single CclECI object that exists on the client.

static CclECI* instance()

Here is an example of its use:

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

listState

Returns a formatted string containing the current state of the object.

const char* listState() const

For example:

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

serverCount

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

unsigned short serverCount() const

In practice, some or all of these servers might not be available. See the information about finding potential servers in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*.

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 the information about finding potential servers in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* for more information.

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 the information about finding potential servers in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* for more information.

CclEPI class

The **CclEPI** 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.

CclEPI constructor

This method initializes the CICS EPI interface on the client.

CclEPI()

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 **CclEPI** constructor returns.

Public methods

diagnose

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

const char* diagnose() const

exCode

Returns an enumeration indicating the most recent exception code.

Deprecated method

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

Ccl::ExCode exCode() const

The possible values are listed under "C++ Exception Objects" on page 106.

exCodeText

Returns a text string describing the most recent exception code.

Deprecated method

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

const char* exCodeText() const

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

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

unsigned short serverCount()

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

Returns an enumeration indicating the state of the EPI.

State state() const

Possible values are:
active EPI has been initialized successfully
discon
EPI has terminated
error EPI initialization has failed

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.

Deprecated method

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

void terminate()

Because the terminate method is started during CclEPI object destruction, you do not need to start 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, subclass the **CclECI** or **CclEPI** class and provide your own implementation of the **handleException** method. See the information about handling exceptions in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*. 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

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

const char* abendCode()

className

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

const char* className() const

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

Returns the exception code.

Ccl::ExCode exCode() const

For further information see "C++ Exception Objects" on page 106.

exCodeText

Returns a text string that describes the exception code.

const char* exCodeText() const

exObject

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

void* exObject() const

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 must be cast to a **CclFlow***. For example:

  ```
  CclFlow* pflo = (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

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

const char* methodName() const

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

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.

Color backgroundColor() const

baseAttribute

Returns the 3270 base attribute of the field.

char baseAttribute() const

column

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

unsigned short column() const

dataTag

Returns an enumeration indicating whether the data in the field has been modified.

BaseMDT dataTag() const

Possible values are:

- modified
- unmodified

foregroundColor

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.

Color foregroundColor() const

highlight

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.

Highlight highlight() const

inputProt

Returns an enumeration indicating whether the field is protected.

BaseProt inputProt() const

Possible values are:

- protect
- unprotect

inputType

Returns an enumeration indicating the input data type for this field.

BaseType inputType() const

Possible values are:

- alphanumeric
- numeric

intensity

Returns an enumeration indicating the field intensity.

BaseInts intensity() const

Possible values are :

- dark
- normal
- intense

length

Returns the total length of the field.

unsigned short length() const

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 85 method.

position

unsigned short position() const

Returns the position of the start of the field on the screen, given by position = column number + (*n* x 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

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

unsigned short row() const

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

Returns the text currently held in the field.

const char* text() const

textLength

Returns the number of characters currently held in the field.

unsigned short textLength() const

transparency

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.

Transparency transparency() const

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.

See the information about compiling and running a C++ Client application in the *CICS Transaction Gateway for Multiplatforms: Developing Applications* 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
Returns the abend code from the most recently executed CICS transaction, or blank if there have been none.

const char* abendCode() const

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

CallType callType() const

callTypeText
Returns the name of the most recent server request.

const char* callTypeText() const

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

CclConn* connection() const

diagnose

const char* diagnose() const

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

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

flowId
Returns the unique identity of this CclFlow object.

unsigned short flowId() const

forceReset
Makes the flow inactive and resets it.

void forceReset()

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 a 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 the information about Programming in C++ in the *CICS Transaction Gateway for Multiplatforms: Developing*

Applications. To deal with replies, subclass **CclFlow** and provide your own implementation of **handleReply**. The default implementation merely returns to the caller.

listState

Returns a formatted string containing the current state of the object.

const char* listState() const

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

Returns an enumeration, defined within the **Ccl** class indicating the type of synchronization being used.

Ccl::Sync syncType() const

Possible values are shown in "Sync" on page 67.

timeout

Retrieves the current timeout value set for the flow object.

short timeout()

uow

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

CclUOW* uow() const

wait

Waits for a reply from the server, blocking the client process in the meantime.

void wait()

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

Returns an enumeration indicating the most recent exception code.

Deprecated method

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

Ccl::ExCode exCode() const

The possible values are listed in "C++ Exception Objects" on page 106.

exCodeText

Returns a text string describing the most recent exception code.

Deprecated method

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

const char* exCodeText() const

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

unsigned short cursorCol() const

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

unsigned short cursorRow() const

depth
Returns the number of rows in the screen.

unsigned short depth() const

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

Returns the number of fields in the screen.

unsigned short fieldCount() const

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.

const char* mapName()

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

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.

const char* mapSetName()

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 might 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 94 enumerations at the end of this section.

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
Returns the number of columns on the screen.

unsigned short width() const

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

CclSecTime* expiryTime() const

invalidCount
Returns the Number of times that an invalid password has been entered for the user ID.

unsigned short invalidCount() const

lastAccessTime
Returns a CclSecTime object that contains the date and time when the user ID was last accessed.

CclSecTime* lastAccessTime() const

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

CclSecTime* lastVerifiedTime() const

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

unsigned short day() const

get_time_t
Returns the date and time in a time_t format.

time_t get_time_t() const

get_tm
Returns the date and time in a tm structure.

tm get_tm() const

hours
Returns the hours with a range from 0 to 23.

unsigned short hours() const

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

unsigned short hundredths() const

minutes
Returns the minutes with a range from 0 to 59.

unsigned short minutes() const

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

unsigned short month() const

seconds
Returns the seconds with a range from 0 to 59.

unsigned short seconds() const

year
Returns a 4–digit year

unsigned short year() const

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. 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
Returns a text description of the last error.

const char* diagnose() const

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
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.

State state() const

terminal

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).

CclTerminal* terminal() const

transID

Returns the 4-letter name of the current transaction.

const char* transID() const

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 CICS 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 start 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 CICS 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:

- `CclTerminal::SignonCapable`

- `CclTerminal::SignonIncapable`

userid
The name of the user ID to associate with this terminal resource

password
The password to associate with the user ID

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

You can call the method before you install a terminal. It changes only the terminal definition; the new user ID and password will be used for the terminal when install is called.

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

userid
The new user ID

password
The new password for *userid*

Allows you to re-define the user ID and password for a terminal resource.

changePassword

Allows a Client application to change the password held in the terminal object and the password recorded by an external security manager for the user ID held in the terminal object.

CclSecAttr* changePassword(const char* newPassword)

newPassword
The new password

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

CCSid

Returns the selected code page as an unsigned short.

unsigned short CCSid()

diagnose

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

const char* diagnose()

disconnect (1)

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

void disconnect()

disconnect (2)

Disconnects the terminal from CICS.

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

Returns the reasons for a disconnection.

void discReason(void)

See "EndTerminalReason" on page 105.

exCode

Returns an enumeration indicating the most recent exception code.

Deprecated method

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

Ccl::ExCode exCode() const

The possible values are listed in "C++ Exception Objects" on page 106.

exCodeText

Returns a text string describing the most recent exception code.

Deprecated method

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

const char* exCodeText() const

install

Connects a non-connected terminal resource.

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.

Throws an invalidState error if already connected, or a timeout error if a timeout occurs.

makeSecurityDefault

Informs the client that the current user ID 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.

void makeSecurityDefault()

netName

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

const char* netName() const

password

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

const char* password()

poll

Polls for data from the CICS server.

Ccl::Bool poll()

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 must be in server state for poll to be called.

A CICS server transaction can send more than one reply in response to a CclTerminal **send** call. More than one CclTerminal **poll** call can therefore be needed

to collect all the replies. Use the CclTerminal **state** method to find out whether further replies are expected. If there are, the value returned will be server. See the information about EPI call synchronization types in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*.

queryATI

Returns an enumeration indicating whether the "Automatic Transaction Initiation" (ATI) is enabled or disabled.

ATIState queryATI()

Possible values are:

- disabled
- enabled

readTimeout

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

const char* readTimeout()

receiveATI

Waits for and receives 3270 data stream for a CICS ATI transaction.

void receiveATI(CclSession* *session***)**

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

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

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

CclScreen* screen() const

send (1)

Formats and sends a 3270 data stream, starting the named transaction.

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.

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)

Formats and sends a 3270 data stream.

void send(CclSession* *session***)**

The *session* parameter is described above.

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

Indicates whether the ATI is to be enabled or disabled.

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

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

signonType signonCapability()

Possible values are:

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

state

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

State state() const

serverName

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

const char* serverName() const

termID

Returns the 4-character terminal ID.

const char* termID() const

transID

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.

const char* transID() const

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

const char* userId()

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

CclSecAttr* verifyPassword()

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 shut down.

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 74).

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

Creates a CclUOW object.

CclUOW()

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
Make this UOW inactive and reset it.

void forceReset()

listState
Returns a zero-terminated formatted string containing the current state of the object.

const char* listState() const

For example:

```
UOW state..&CclUOW=0004899C  &CclUOWI=00203BD0
&CclConnI=00000000  uowId=0  &CclFlowI=00000000
```

uowId
Returns the identifier of the UOW. 0 means that the UOW is either complete or has not yet started and it is, therefore, inactive.

unsigned long uowId() const

C++ Exception Objects

Exception objects for the C++ classes.

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 2. 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	

Table 2. Exception codes (continued)

Enumeration	Text	Description	ECI	EPI
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
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
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

Table 2. Exception codes (continued)

Enumeration	Text	Description	ECI	EPI
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. 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. 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 start the method, e.g. terminal object still in server state and an attempt to send data is made.	Yes	Yes
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 user ID is null.	Yes	Yes
Ccl::nullNewPassword	A NULL new password was supplied	The provided password is null.	Yes	Yes

Table 2. Exception codes (continued)

Enumeration	Text	Description	ECI	EPI
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 user ID is unknown.	Yes	Yes
Ccl:invalidTermid	Termid is invalid	The terminal ID is invalid.		Yes
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

Chapter 4. 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

The interfaces available for Visual Basic.

The interfaces are:

```
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

Ways of creating an object.

To create an object use one of the following:

```
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

Methods available on this class.

AppendString

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

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

string
The source string.

Data

Returns the contents of the buffer as a Byte array.

Data() as Variant

ExtractString

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

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.

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

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

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.

Length

Returns the length of the data area in bytes.

Length() as Integer

Overlay

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

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.

SetData

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

SetData(*array* **as Variant)**

array
The array containing the source data.

SetLength

Changes the current length of the data area.

SetLength(*length* **as Integer)**

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

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

Copies the supplied string into the object.

SetString(*string* **as String)**

string
Source string

String

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

String() as 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

The interfaces available for Visual Basic.

The interfaces are:

```
Dim var as Object
Dim var as CclOConn
```

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

Ways of creating an object.

To create an object use one of the following:

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

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

Methods

Methods available on this class.

AlterSecurity

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

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

newUserid
The new user ID

newPassword
The new password corresponding to the new user ID.

Cancel

Cancels **Changed** calls that were previously issued to the server associated with this connection.

Cancel(*flow* **as Object)**

Cancel(*flow* **as CclOFlow)**

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

Changed

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.

Changed(*flow* **as Object)**

Changed(*flow* **as CclOFlow)**

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

ChangePassword

Allows a client application to change the password held in the Connect object and the password recorded by an external security manager for the user ID held in the Connect object.

ChangePassword (*newPassword* **as String) as Object**

ChangePassword (*newPassword* **as String) as CclOSecAttr**

newPassword
The new password

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

Supplies details of the CICS server.

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.

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

Calls the specified program on the server.

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

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 is 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 is supplied if no UOW is to be used.

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

MakeSecurityDefault

Informs the client that the current user ID 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.

MakeSecurityDefault()

Password

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

Password() as String

ServerName

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

ServerName() as String

ServerStatus

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

ServerStatus() as Integer

ServerStatus() as CclConnectStatusCodes

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

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

ServerStatusText() as String

Status

Requests the status of the server connection.

Status(*flow* **as Object)**

Status(*flow* **as CclOFlow)**

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

TranDetails

Supplies additional information about the related transaction to the CICS server.

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.

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 115.

UnpaddedPassword

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

UnpaddedPassword() as String

UnpaddedServerName

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

UnpaddedServerName() as String

UnpaddedUserid

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

UnpaddedUserid() as String

UserId

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

UserId() as String

VerifyPassword

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 user ID held in the Connect object.

VerifyPassword() as Object

VerifyPassword() as CclOSecAttr

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

The interfaces available for Visual Basic.

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

Ways of creating an object.

You can create an object in two ways:

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

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

Methods

Methods available on this class.

ErrorFormat

Returns a value indicating the current setting for the Error Message Format.

ErrorFormat() as Integer

See "SetErrorFormat" on page 120 for a current list of valid values.

ErrorOffset

Returns a value that can be used to convert a Client daemon error value retrieved from the ERR.Number method into the documented ExCode error values.

ErrorOffset() as Long

For more information on how to do this, see the information about handling exceptions in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*.

ErrorWindow

Determines whether or not an error window is displayed to the user. This is a deprecated method.

Deprecated method

: Do not use this method in a 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

Returns an enumeration that indicates the last ECI error. This is a deprecated method.

Deprecated method

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

ExCode() as Integer

ExCode() as CclECIExceptionCodes

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
Returns a text string describing the last ECI error.

Deprecated method

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

ExCodeText() as String

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

ServerCount() as Integer

ServerDesc
Returns the description of the *index*th server.

ServerDesc(*index* **as Integer) as String**

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

ServerName
Returns the name of the *index*th server.

ServerName(*index* **as Integer) as String**

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

SetErrorFormat
Specifies the error message format.

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.

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 can be used. You create a CclOEPI object before you create CclOTerminal objects to connect to CICS servers. The **Diagnose**, **ExCode**, and **State** methods provide information on error conditions.

Interface Selection

The interfaces available for Visual Basic.

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

```
Dim var as Object
Dim var as CclOEPI
```

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

Ways of creating an object.

You can create an object in two ways

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

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

Methods

Methods available on this class.

Diagnose

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

Diagnose() as String

ErrorFormat

Returns a value indicating the current setting for the Error Message Format.

ErrorFormat() as Integer

See “SetErrorFormat” on page 123 for a current list of valid values.

ErrorOffset

Returns a value that can be used to convert a Client daemon error value retrieved from the ERR.Number method into the documented ExCode error values.

ErrorOffset() as Long

For more information on how to do this, see the information about handling exceptions in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*.

ErrorWindow

Determines whether or not an error window is displayed to the user. This is a deprecated method.

Deprecated method

: Do not use this method in a 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

Returns the condition code. This is a deprecated method.

Deprecated method

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

ExCode() as Integer

ExCode() as CclEPIExceptionCodes

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

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

Deprecated method

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

ExCodeText() as String

ServerCount

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

ServerCount() as Integer

ServerDesc

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.

ServerDesc(*index* **as Integer) as String**

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

ServerName

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.

ServerName(*index* **as Integer) as String**

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

SetErrorFormat

Specifies the format for error messages.

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.

State

Returns a value that indicates the state of the EPI.

State() as Integer

State() as CclEPIStates

Possible values are:

cclActive
Initialized

cclDiscon
Terminated

cclError
Error. See the information about Programming in COM in the *CICS Transaction Gateway for Multiplatforms: Developing Applications*.

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

Terminate

Terminates the Client daemon EPI in a controlled manner.

Deprecated method

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

Terminate()

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

The interfaces available for Visual Basic.

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

Methods available on this class.

AppendText

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

AppendText(*textString* **as String)**

textString
The text string to be appended to the field.

BackgroundColor

Returns a value which indicates the background color of the field.

BackgroundColor() as Integer

BackgroundColor() as CclColorAttributes

Returns a value which indicates the background color of the field as listed in "CclField Color Attributes" on page 149.

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

BaseAttribute

Returns the 3270 base attribute of the field.

BaseAttribute() as Integer

Column

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

Column() as Integer

DataTag

Returns a value which indicates whether the data in the field has been modified.

DataTag() as Integer

DataTag() as CclModifiedAttributes

Possible values are:

- cclModified
- cclUnmodified

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

ForegroundColor

Returns a value which indicates the foreground color of the field.

ForegroundColor() as Integer

ForegroundColor() as CclColorAttributes

Returns a value which indicates the foreground color of the field as listed in "CclField Color Attributes" on page 149.

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

Highlight

Returns a value which indicates the type of highlight used.

Highlight() as Integer

Highlight() as CclHighlightAttributes

Returns a value which indicates which type of highlight is being used as listed in "CclField Highlight Attributes" on page 149.

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

InputProt

Returns a value which indicates whether the field is protected.

InputProt() as Integer

InputProt() as CclProtAttributes

Possible values are:

- cclProtect
- cclUnprotect

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

InputType

Returns a value which indicates whether the field is alphanumeric or numeric.

InputType() as Integer

InputType() as CclNumericAttributes

Possible values are:

- cclAlphanumeric
- cclNumeric

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

Intensity

Returns a value which indicates whether the field is normal, intense or dark.

Intensity() as Integer

Intensity() as CclIntensityAttributes

Possible values are:

- cclDark
- cclNormal
- cclIntense

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

Length

Returns the total length of the field.

Length() as Integer

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 128 method.

Position

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; and so on.

Position() as Integer

ResetDataTag

Resets the modified data tag (MDT) to cclUnmodified.

ResetDataTag()

Row

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

Row() as Integer

SetBaseAttribute

Sets the 3270 base attribute.

SetBaseAttribute(*Attribute* **as Integer)**

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

SetExtAttribute

Sets the extended 3270 attribute.

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

Attribute
The type of extended attribute to be set.

Value
The value of the extended attribute.

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

SetText

Copies *textString* into the field.

SetText(*textString* **as String)**

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

Text

Returns the text currently held in the field.

Text() as String

TextLength

Returns the number of characters currently held in the field.

TextLength() as Integer

Transparency

Returns a value which indicates the background transparency of the field.

Transparency() as Integer

Transparency() as CclTransparencyAttributes

Returns a value which indicates the background transparency of the field as listed in "CclField Transparency Attributes" on page 149.

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

The interfaces available for Visual Basic.

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

Ways of creating an object.

You can create an object in two ways:

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

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

Methods

Methods available on this class.

AbendCode

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

AbendCode() as String

CallType

Returns the call type for the call currently being executed.

CallType() as Integer

CallType() as CclFlowCallTypes

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

CallTypeText

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

CallTypeText() as String

Diagnose

Returns text describing the current state of the flow object.

Diagnose() as String

Flowid

Returns a unique identifier for this flow object.

Flowid() as Integer

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.

ForceReset()

Poll

Indicates whether a reply has been received from a deferred synchronous **Backout**, **Cancel**, **Changed**, **Commit**, **Link**, or **Status** call request.

Poll(*commArea* **as Object) as Boolean**

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 you do not want a COMMAREA to be returned.

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
Sets the synchronization type required for this CclOFlow object.

SetSyncType(*syncType* **as Integer)**

SetSyncType(*syncType* **as CclFlowSyncTypes)**

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

- cclSync
- cclDSyncCclDSync

If cclSync is used, **link** and **status** calls using this flow block the calling program until a reply is received from CICS. If cclDSyncCclDSync 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
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.

SetTimeout(Timeout as Integer)

SyncType
Returns the type of synchronization being used.

SyncType() as Integer

SyncType() as CclFlowSyncTypes

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

Timeout
Returns the current timeout value set for the flow object.

Timeout() as Integer

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.

Wait()

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

The interfaces available for Visual Basic.

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

```
Dim var as Object
Dim var as CclOMap
```

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

Ways of creating an object.

You can create an object in two ways:

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

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

Methods

Methods available on this class.

ExCode

Returns a value that indicates the current condition code.

Deprecated method

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

ExCode() as Integer

ExCode() as CclEPIExceptionCodes

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

FieldByName

Returns the specified CclOField object.

FieldByName(*name* **as Integer) as Object**

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.

Validate

Validates a BMS map against the current screen and verifies that a specific map has been received from the CICS server.

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

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.

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

The interfaces available for Visual Basic.

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

Methods available on this class.

CursorCol

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

CursorCol() as Integer

CursorRow

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

CursorRow() as Integer

Depth

Returns the number of rows on the screen.

Depth() as Integer

FieldByIndex

Returns the index number of the field required.

FieldByIndex(*index* **as Integer) as Object**

FieldByIndex(*index* **as Integer) as CclOField**

index
The first field is number 1.

FieldByPosition

Returns the row number and column number of the field.

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

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

Returns the number of fields on the screen.

FieldCount() as Integer

MapName

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.

MapName() as String

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

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.

MapSetName() as String

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 might 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

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

SetAID(*key* **as Integer)**

SetAID(*key* **as CclADIKeys)**

key

The AID key value as listed in "CclScreen AID key codes" on page 147.

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

SetCursor

Sets the column number and row number of the cursor.

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

Returns the number of columns on the screen.

Width() as Integer

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

The interfaces available for Visual Basic.

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

```
Dim var as Object
Dim var as CclOSecAttr
```

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

Methods available on this class.

ExpiryTime

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

ExpiryTime() as Object

ExpiryTime() as CclOSecTime

InvalidCount

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

InvalidCount() as Integer

LastAccessTime

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

LastAccessTime() as Object

LastAccessTime() as CclOSecTime

LastVerifiedTime

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

LastVerifiedTime() as Object

LastVerifiedTime() as CclOSecTime

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

The interfaces available for Visual Basic.

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

Methods available on this class.

Day

Returns the day in the range 1 to 31.

unsigned short Day() as Integer

GetDate

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

GetDate() as Date

Hours

Returns the hours in the range 0 to 23.

unsigned short Hours() as Integer

Hundredths

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

unsigned short Hundredths() as Integer

Minutes

Returns the minutes in the range 0 to 59.

unsigned short Minutes() as Integer

Month

Returns the month in the range 1 to 12.

unsigned short Month() as Integer

Seconds

Returns the seconds in the range 0 to 59.

unsigned short Seconds() as Integer

Year

Returns a 4 digit year.

unsigned short Year() as Integer

Session COM class

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

Interface Selection

The interfaces available for Visual Basic.

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

```
Dim var as Object
Dim var as CclOSession
```

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

Ways of creating an object.

You can create an object in two ways:

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

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

Methods

Methods available on this class.

Diagnose

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

Diagnose() as String

SetSyncType

Sets the synchronization type required for this CclOSession object.

SetSyncType(*syncType* **as Integer)**

SetSyncType(*syncType* **as CclFlowSyncTypes)**

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

- cclSync
- cclDSyncCclDSync

If cclSync is used, **Start** and **Send** calls using this flow will block the calling program until a reply is received from CICS. If cclDSyncCclDSync 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

Returns a value which indicates the current state of the session.

State() as Integer

State() as CclEPIStates

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
Returns the four-letter name of the current transaction.

TransId() as String

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

The interfaces available for Visual Basic.

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

```
Dim var as Object
Dim var as CclOTerminal
```

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

Ways of creating an object.

You can create an object in two ways:

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

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

Methods

Methods available on this class.

AlterSecurity
Redefines the user ID and password for a terminal resource that has been constructed without these values (a sign-on incapable terminal).

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

newPassword
The new password to be given to *newUserid*.

newUserid
The new user ID.

This method can be called before you install a terminal. It changes the terminal definition; the new user ID and password are be used for the terminal when install is called.

CCSId

Returns a long showing the selected code page.

CCSId() as long

ChangePassword

Enables a client application to change the password held in the terminal object and the password recorded by an external security manager for the user ID held in the terminal object.

ChangePassword(*newPassword* **as String) as Object**

ChangePassword(*newPassword* **as String) as CclOSecAttr**

newPassword
The new password to be given

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

Establishes a 3270 communication to the specified CICS server.

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 CICS 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.

Devtype

Returns the terminal device type as a string.

Devtype() as String

Diagnose

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

Diagnose() as String

Disconnect

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

Disconnect()

DisconnectWithPurge

DisconnectWithPurge disconnects the terminal from CICS and attempts to purge all outstanding running transactions. This purge function does not cancel ATI requests queued against the terminal.

DisconnectWithPurge()

DiscReason

This method will return an enumeration showing the reason the terminal has been disconnected.

DiscReason() as CclEndTermReasons

Possible values are shown in "CclTerminal EndTermReasons" on page 147.

ExCode

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

Deprecated method

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

ExCode() as Integer

ExCode() as CclEPIExceptionCodes

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

ExCodeText

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

Deprecated method

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

ExCodeText() as String

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

Install

Installs a non-connected terminal resource.

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

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.

A `cclInvalidState` error is raised if the terminal is already installed.

MakeSecurityDefault

Informs the client that the current user ID 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.

MakeSecurityDefault()

NetName

Returns the network name of the terminal.

NetName() as String

Password

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

Password() as String

Poll

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

Poll() as Boolean

Possible values are:
True No further replies outstanding
False Further replies outstanding

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

PollForReply

Checks to see whether replies have been received from a deferred synchronous Start or Send request.

PollForReply() as Boolean

Possible values are

true Replies have been received

false No replies have been received

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

QueryATI

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

QueryATI() as Integer

QueryATI() as CclATIStates

Possible values are:

- cclATIEnabled
- cclATIDisabled

ReadTimeout

Returns the read timeout setting for the terminal.

ReadTimeout() as Integer

ReceiveATI

Waits for and receives 3270 data stream for a CICS ATI transaction.

ReceiveATI (*session* **as Object)**

ReceiveATI (*session* **as CclOSession)**

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

The CclOSession object supplied can only be synchronous.

Screen

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

Screen() as Object

Send

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

Send(*session* **as Object)**

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.

ServerName

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

ServerName() as String

SetATI

Indicates whether the ATI is to be enabled or disabled.

SetATI(*stateVal* **as Integer)**

SetATI(*stateVal* **as CclATIStates)**

stateVal
Possible values are:

- cclATIEnabled
- cclATIDisabled

SetTermDefns

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

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 CICS 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 user ID to associate with this terminal resource.

password
The password to associate with the user ID.

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.

SignonCapability

Returns the type of terminal installed.

SignonCapability() as Integer

SignonCapability() as CclSignonTypes

Possible values are:

- cclSignonCapable
- cclSignonIncapable

Start

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

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

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.

State
Returns a value which indicates the current state of the session.

State() as Integer

State() as CclEPIStates

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
Returns the terminal ID.

TermId() as String

TransId
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.

TransId() as String

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

Userid() as String

VerifyPassword
Enables a client application to verify that the password held in the terminal object matches the password recorded by an external security manager for the user ID held in the terminal object.

VerifyPassword() as Object

VerifyPassword() as CclOSecAttr

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 115).

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 "COM Error Code References" on page 151.

COM EPI Specific Constants

Synchronization Types

Table 3. Synchronization types

VB Enumeration	Value	Description
cclSync	0	Synchronous call type
cclDsync	1	Deferred synchronous call type

CclEPI states

Table 4. CclEPI States

VB Enumeration	Value	Description
cclEPIActive	0	EPI initialized
cclDiscon	1	EPI Terminated
cclEPIError	2	EPI failed to initialize, handle exception for more information

CclSession States

Table 5. 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
ccISessionDiscon	3	Disconnected
ccISessionError	4	Session Error, handle exception for more information

CclTerminal States

Table 6. CclTerminal States

VB Enumeration	Value	Description
cclInit	0	Terminal defined but not installed
cclActive	1	Terminal connected (not used)

Table 6. CclTerminal States (continued)

VB Enumeration	Value	Description
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 7. CclTerminal ATI states

VB Enumeration	Value	Description
cclATIEnabled	0	ATIs are allowed
cclATIDisabled	1	ATIs are not allowed

CclTerminal EndTermReasons

Table 8. 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 9. CclTerminal Sign-on Types

VB Enumeration	Value	Description
cclSignonCapable	0	Terminal supports sign-on transaction
cclSignonIncapable	1	Terminal does not support sign-on transaction
cclSignonUnknown	2	Terminal sign-on capability is unknown

CclScreen AID key codes

Table 10. 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

Table 10. CclScreen AID key codes (continued)

VB Enumeration	Value	Description
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
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 11. CclField Protected state attributes

VB Enumeration	Value	Description
cclProtect	0	Protected Field (cannot be modified)
cclUnprotect	1	Unprotected (input) field

CclField Numeric Attributes

Table 12. CclField Numeric Attributes

VB Enumeration	Value	Description
cclAlphanumeric	0	Alphanumeric input field
cclNnumeric	1	Numeric input field

CclField Intensity Attributes

Table 13. CclField Intensity attributes

VB Enumeration	Value	Description
cclNormal	0	Normal display
cclIntense	1	Intensified display
cclDark	2	Non-display field

CclField Modified Attributes

Table 14. CclField Modified Attributes

VB Enumeration	Value	Description
cclUnmodified	0	Field has not been changed
cclModified	1	Field has been changed

CclField Highlight Attributes

Table 15. 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 16. CclField Transparency attributes

VB Enumeration	Value	Description
cclTrnDefault	0	Default (opaque) field background
cclTrnOr	1	Transparent field background (OR)
cclTrnXorcclTrnXor	2	Transparent field background (XOR)
cclTrnOpaque	3	Opaque field background

CclField Color Attributes

Table 17. CclField Color attributes

VB Enumeration	Value	Description
cclDefaultColor	0	
cclBlue	1	
cclRed	2	
cclPink	3	
cclGreen	4	
cclCyan	5	

Table 17. CclField Color attributes (continued)

VB Enumeration	Value	Description
cclYellow	6	
cclNeutral	7	
cclBlack	8	
cclDarkBlue	9	
cclOrange	10	
cclPurple	11	
cclPaleGreen	12	
cclPaleCyan	13	
cclGray	14	
cclWhite	15	

COM ECI Constants

Synchronization Types

Table 18. Synchronization types

VB Enumeration	Value	Description
cclSync	0	Synchronous call type
cclDsync	1	Deferred synchronous call type

Flow status types

Table 19. 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 20. 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

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 ended abnormally	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
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. 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. 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	

Enumeration	Value	Description	ECI	EPI
cclParameter	22	Incorrect parameter supplied	Yes	Yes
cclInvalidState	23	The Object is not in the correct state to start 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
cclNoUserid	33	The object's user ID 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 user ID 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

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 “COM Error Code References” on page 151.

COM EPI Specific Constants

Synchronization Types

Table 21. Synchronization types

VB Enumeration	Value	Description
cclSync	0	Synchronous call type
cclDsync	1	Deferred synchronous call type

CclEPI states

Table 22. CclEPI States

VB Enumeration	Value	Description
cclEPIActive	0	EPI initialized
cclDiscon	1	EPI Terminated
cclEPIError	2	EPI failed to initialize, handle exception for more information

CclSession States

Table 23. CclSession States

VB Enumeration	Value	Description
cclSessionIdle	0	Idle, client needs to initiate transaction
cclSessionServer	1	Waiting for server
cclSessionClient	2	Waiting for Client daemon to respond
cclSessionDiscon	3	Disconnected
cclSessionError	4	Session Error, handle exception for more information

CclTerminal States

Table 24. CclTerminal States

VB Enumeration	Value	Description
cclInit	0	Terminal defined but not installed
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 25. CclTerminal ATI states

VB Enumeration	Value	Description
cclATIEnabled	0	ATIs are allowed

Table 25. CclTerminal ATI states (continued)

VB Enumeration	Value	Description
cclATIDisabled	1	ATIs are not allowed

CclTerminal EndTermReasons

Table 26. 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 27. CclTerminal Sign-on Types

VB Enumeration	Value	Description
cclSignonCapable	0	Terminal supports sign-on transaction
cclSignonIncapable	1	Terminal does not support sign-on transaction
cclSignonUnknown	2	Terminal sign-on capability is unknown

CclScreen AID key codes

Table 28. 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

Table 28. CclScreen AID key codes (continued)

VB Enumeration	Value	Description
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
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 29. CclField Protected state attributes

VB Enumeration	Value	Description
cclProtect	0	Protected Field (cannot be modified)
cclUnprotect	1	Unprotected (input) field

CclField Numeric Attributes

Table 30. CclField Numeric Attributes

VB Enumeration	Value	Description
cclAlphanumeric	0	Alphanumeric input field
cclNnumeric	1	Numeric input field

CclField Intensity Attributes

Table 31. CclField Intensity attributes

VB Enumeration	Value	Description
cclNormal	0	Normal display
cclIntense	1	Intensified display
cclDark	2	Non-display field

CclField Modified Attributes

Table 32. CclField Modified Attributes

VB Enumeration	Value	Description
cclUnmodified	0	Field has not been changed
cclModified	1	Field has been changed

CclField Highlight Attributes

Table 33. 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 34. CclField Transparency attributes

VB Enumeration	Value	Description
cclTrnDefault	0	Default (opaque) field background
cclTrnOr	1	Transparent field background (OR)
cclTrnXorcclTrnXor	2	Transparent field background (XOR)
cclTrnOpaque	3	Opaque field background

CclField Color Attributes

Table 35. 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	
cclDarkBlue	9	
cclOrange	10	
cclPurple	11	
cclPaleGreen	12	
cclPaleCyan	13	
cclGray	14	
cclWhite	15	

COM ECI Constants

Synchronization Types

Table 36. Synchronization types

VB Enumeration	Value	Description
cclSync	0	Synchronous call type
cclDsync	1	Deferred synchronous call type

Flow status types

Table 37. 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 38. 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

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

Enumeration	Value	Description	ECI	EPI
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 ended abnormally	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
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. 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. 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 start 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

Enumeration	Value	Description	ECI	EPI
cclTimeout	31	Timeout occurred before response from Server	Yes	Yes
cclNoPassword	32	The object's password is null.	Yes	Yes
cclNoUserid	33	The object's user ID 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 user ID 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

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 “COM Error Code References” on page 151.

COM EPI Specific Constants

Synchronization Types

Table 39. Synchronization types

VB Enumeration	Value	Description
cclSync	0	Synchronous call type
cclDsync	1	Deferred synchronous call type

CclEPI states

Table 40. CclEPI States

VB Enumeration	Value	Description
cclEPIActive	0	EPI initialized
cclDiscon	1	EPI Terminated
cclEPIError	2	EPI failed to initialize, handle exception for more information

CclSession States

Table 41. CclSession States

VB Enumeration	Value	Description
cclSessionIdle	0	Idle, client needs to initiate transaction
cclSessionServer	1	Waiting for server
cclSessionClient	2	Waiting for Client daemon to respond
cclSessionDiscon	3	Disconnected
cclSessionError	4	Session Error, handle exception for more information

CclTerminal States

Table 42. CclTerminal States

VB Enumeration	Value	Description
cclInit	0	Terminal defined but not installed
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 43. CclTerminal ATI states

VB Enumeration	Value	Description
cclATIEnabled	0	ATIs are allowed
cclATIDisabled	1	ATIs are not allowed

CclTerminal EndTermReasons

Table 44. 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

Table 44. CclTerminal ATI states (continued)

VB Enumeration	Value	Description
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 45. CclTerminal Sign-on Types

VB Enumeration	Value	Description
cclSignonCapable	0	Terminal supports sign-on transaction
cclSignonIncapable	1	Terminal does not support sign-on transaction
cclSignonUnknown	2	Terminal sign-on capability is unknown

CclScreen AID key codes

Table 46. 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

Table 46. CclScreen AID key codes (continued)

VB Enumeration	Value	Description
cclPF19	23	Program Function key 19
cclPF20	24	Program Function key 20
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 47. CclField Protected state attributes

VB Enumeration	Value	Description
cclProtect	0	Protected Field (cannot be modified)
cclUnprotect	1	Unprotected (input) field

CclField Numeric Attributes

Table 48. CclField Numeric Attributes

VB Enumeration	Value	Description
cclAlphanumeric	0	Alphanumeric input field
cclNnumeric	1	Numeric input field

CclField Intensity Attributes

Table 49. CclField Intensity attributes

VB Enumeration	Value	Description
cclNormal	0	Normal display
cclIntense	1	Intensified display
cclDark	2	Non-display field

CclField Modified Attributes

Table 50. CclField Modified Attributes

VB Enumeration	Value	Description
cclUnmodified	0	Field has not been changed
cclModified	1	Field has been changed

CclField Highlight Attributes

Table 51. 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

Table 51. CclField Highlight attributes (continued)

VB Enumeration	Value	Description
cclHltReverse	3	Reverse video text
cclHltUnderscore	4	Underscored text
cclHltIntense	5	High intensity text

CclField Transparency Attributes

Table 52. CclField Transparency attributes

VB Enumeration	Value	Description
cclTrnDefault	0	Default (opaque) field background
cclTrnOr	1	Transparent field background (OR)
cclTrnXorcclTrnXor	2	Transparent field background (XOR)
cclTrnOpaque	3	Opaque field background

CclField Color Attributes

Table 53. 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	
cclDarkBlue	9	
cclOrange	10	
cclPurple	11	
cclPaleGreen	12	
cclPaleCyan	13	
cclGray	14	
cclWhite	15	

COM ECI Constants

Synchronization Types

Table 54. Synchronization types

VB Enumeration	Value	Description
cclSync	0	Synchronous call type
cclDsync	1	Deferred synchronous call type

Flow status types

Table 55. 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 56. 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

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	

Enumeration	Value	Description	ECI	EPI
cclTransaction	13	ECI program ended abnormally	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
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. 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. 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 start 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
cclNoUserid	33	The object's user ID 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

Enumeration	Value	Description	ECI	EPI
cclPasswordRejected	39	Change password failed because the password doesn't conform to standards defined	Yes	Yes
cclUseridInvalid	40	The user ID 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

Interface Selection

The interfaces available for Visual Basic.

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

Ways of creating an object.

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

Methods available on this class.

BackOut

Terminates this UOW and backs out all changes made to recoverable resources in the server.

BackOut(*flow* **as Object)**

BackOut(*flow* **as CclOFlow)**

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

Commit

Terminates this UOW and commits all changes made to recoverable resources in the server.

Commit(*flow* as Object)

Commit(*flow* as CclOFlow)

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

ForceReset

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

ForceReset()

UowId

Returns the identifier of the UOW. A zero return indicates that the UOW is either complete or has not yet started, and is therefore inactive.

UowId() as long

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 "COM Error Code References" on page 151.

COM EPI Specific Constants

Synchronization Types

Table 57. Synchronization types

VB Enumeration	Value	Description
cclSync	0	Synchronous call type
cclDsync	1	Deferred synchronous call type

CclEPI states

Table 58. CclEPI States

VB Enumeration	Value	Description
cclEPIActive	0	EPI initialized
cclDiscon	1	EPI Terminated
cclEPIError	2	EPI failed to initialize, handle exception for more information

CclSession States

Table 59. CclSession States

VB Enumeration	Value	Description
cclSessionIdle	0	Idle, client needs to initiate transaction
cclSessionServer	1	Waiting for server
cclSessionClient	2	Waiting for Client daemon to respond
cclSessionDiscon	3	Disconnected
cclSessionError	4	Session Error, handle exception for more information

CclTerminal States

Table 60. CclTerminal States

VB Enumeration	Value	Description
cclInit	0	Terminal defined but not installed
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 61. CclTerminal ATI states

VB Enumeration	Value	Description
cclATIEnabled	0	ATIs are allowed
cclATIDisabled	1	ATIs are not allowed

CclTerminal EndTermReasons

Table 62. 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 63. CclTerminal Sign-on Types

VB Enumeration	Value	Description
cclSignonCapable	0	Terminal supports sign-on transaction
cclSignonIncapable	1	Terminal does not support sign-on transaction
cclSignonUnknown	2	Terminal sign-on capability is unknown

CclScreen AID key codes

Table 64. 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
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 65. CclField Protected state attributes

VB Enumeration	Value	Description
cclProtect	0	Protected Field (cannot be modified)
cclUnprotect	1	Unprotected (input) field

CclField Numeric Attributes

Table 66. CclField Numeric Attributes

VB Enumeration	Value	Description
cclAlphanumeric	0	Alphanumeric input field
cclNnumeric	1	Numeric input field

CclField Intensity Attributes

Table 67. CclField Intensity attributes

VB Enumeration	Value	Description
cclNormal	0	Normal display
cclIntense	1	Intensified display
cclDark	2	Non-display field

CclField Modified Attributes

Table 68. CclField Modified Attributes

VB Enumeration	Value	Description
cclUnmodified	0	Field has not been changed
cclModified	1	Field has been changed

CclField Highlight Attributes

Table 69. 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 70. CclField Transparency attributes

VB Enumeration	Value	Description
cclTrnDefault	0	Default (opaque) field background
cclTrnOr	1	Transparent field background (OR)
cclTrnXorcclTrnXor	2	Transparent field background (XOR)
cclTrnOpaque	3	Opaque field background

CclField Color Attributes

Table 71. 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	
cclDarkBlue	9	
cclOrange	10	
cclPurple	11	
cclPaleGreen	12	
cclPaleCyan	13	
cclGray	14	
cclWhite	15	

COM ECI Constants

Synchronization Types

Table 72. Synchronization types

VB Enumeration	Value	Description
cclSync	0	Synchronous call type
cclDsync	1	Deferred synchronous call type

Flow status types

Table 73. Flow status types

VB Enumeration	Value	Description
cclInactive	0	Flow is inactive

Table 73. Flow status types (continued)

VB Enumeration	Value	Description
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 74. 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

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 ended abnormally	Yes	
cclSystemError	14	Unknown internal error occurred	Yes	Yes

Enumeration	Value	Description	ECI	EPI
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
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. 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. 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 start 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
cclNoUserid	33	The object's user ID 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

Enumeration	Value	Description	ECI	EPI
cclUseridInvalid	40	The user ID 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

Chapter 5. Exits

These topics describe the data available to exits.

ECI Client API exits

The ECI Client API exits are available for use with ECI requests that are sent to servers connected by the TCP/IP and SNA protocols. They are not available if using the IPIC protocol.

The exits are called from the Gateway daemon process when running in remote mode and from the application process when running in local mode. The exits are called for ECI requests issued from all APIs. For more information on creating and deploying user exits see ECI and EPI C exits.

Table 75 summarizes the exit names, the parameters passed to each exit, and the possible return codes.

Table 75. Summary of ECI exits

Function name	Parameters	Return codes:
"CICS_EciInitializeExit" on page 176	**Version** **Anchor**	CICS_EXIT_OK CICS_EXIT_NO_EXIT CICS_EXIT_CANT_INIT_EXITS user-defined
"CICS_EciTerminateExit" on page 177	**Anchor**	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_STORAGE user-defined
"CICS_EciExternalCallExit1" on page 178	**Anchor** **Token** **ParmPtr**	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
"CICS_EciExternalCallExit2" on page 179	**Anchor** **Token** **ParmPtr**	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
"CICS_EciSystemIdExit" on page 180	**Anchor** **Token** **ParmPtr** **Reason**	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM CICS_EXIT_GIVE_UP user_defined
"CICS_EciDataSendExit" on page 181	**Anchor** **Token**	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined

Table 75. Summary of ECI exits (continued)

Function name	Parameters	Return codes:
"CICS_EciDataReturnExit" on page 182	**Anchor** **Token** **ParmPtr**	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined
"CICS_EciSetProgramAliasExit" on page 182	**Anchor** **EciParms** **Program**	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined

Identification token

So that the exits can correlate 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. **CICS_EciExternalCallExit1** and **CICS_EciExternalCallExit2** are not called for a reply solicitation request.

The token is unique within the CICS Transaction Gateway instance for the duration of the request. It can be reused when the last exit for the request has been called.

In the case of an extended logical unit of work, the token might be different on different requests within the logical unit of work. Because reuse of the token, and a new program link call cannot be made until the ECI_GET_REPLY request for the previous asynchronous request has completed, the token might also be the same.

The token is 8 bytes long. A value of 8 null bytes is not valid for the token and is not supplied to the exits.

CICS_EciInitializeExit

This exit allows the user to set up an ECI exit initialization environment.

Function name:	Parameters
CICS_EciInitializeExit	Version Anchor

When called

The CICS_EciInitializeExit exit is invoked when the first ECI request is flowed through the CICS Transaction Gateway for remote mode applications, and when the first ECI request is flowed from each local mode application process. The exit is called after ECI 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 is 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

This exit allows the user to clean up the exit environment.

Function name:	**Parameters**
CICS_EciTerminateExit	Anchor

CICS_EciTerminateExit is not called by CICS Transaction Gateway.

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

This exit enables the user to specify the destination CICS server where the ECI request is to be sent.

Function name:	Parameters:
CICS_EciExternalCallExit1	Anchor Token ParmPtr

The exit can 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.

When called

This exit is called once on each program link and each status information call, after the ECI has validated the parameters. It is not called on a reply solicitation call. Although the exit is called when **eci_luw_token** is not zero, any change made to **eci_system_name** in the ECI parameter block is ignored, as the server was selected when the logical unit of work was started. If **eci_system_name** is changed to contain binary zeros as the server name, then CICS Transaction Gateway dynamically selects the server to which the ECI request is sent.

Parameters

Anchor
Input parameter. The pointer setup 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 can 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.

CICS_EciExternalCallExit2

This exit enables the user to view the results of synchronous ECI calls, and is used for information gathering purposes.

Function name:	**Parameters:**
CICS_EciExternalCallExit2	Anchor Token ParmPtr

When called

This exit is called once on every application program link or status information call. It is not called on reply solicitation calls. The exit is 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 setup 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

This exit enables the user to supply a new system ID, if the value supplied in the ECI parameter block is not valid.

Function name:	**Parameters:**
CICS_EciSystemIdExit	Anchor Token ParmPtr Reason

When called

This exit is called when an error occurs that can be corrected by selection of a new system, user ID, 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 can be called 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 setup 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 can 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 is not called, nor is **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

This exit enables the user to time ECI calls, and is used for performance analysis purposes.

Function name:	**Parameters:**
CICS_EciDataSendExit	Anchor Token

When called

As close as possible to the time that the request is sent to the server.

Parameters

Anchor
Input parameter. The pointer setup 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

This exit enables the user to time ECI calls, and is used for performance analysis purposes.

Function name:	Parameters:
CICS_EciDataReturnExit	Anchor Token ParmPtr

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 setup 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

This exit allows the user to change the program name that the Workload Manager of CICS Transaction Gateway for Windows uses for load balancing.

Function name:	**Parameters:**
CICS_EciSetProgramAliasExit	Anchor EciParms Program

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 setup by **CICS_EciInitializeExit**.

ECIParms
ECI parameter block.

Program
The alias name of the ECI program that the Workload Manager uses 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.

EPI Client API exits

The EPI Client API exits can be used with EPI requests sent to CICS servers over either the TCP/IP or SNA protocol.

For more information on creating and deploying user exits see ../progguide/topics/eiexits.dita.

The C EPI exits are:

- 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 76 summarizes the exit names, the parameters passed to each exit, and the possible return codes.

Table 76. 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**	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined

Table 76. Summary of EPI exits (continued)

Function name	Parameters	Return codes:
CICS_EpiReplyExit	**Anchor** **TermIndex** **Data** **Size**	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined
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

This exit enables the user to set up an EPI exit initialization environment.

Function name:	Parameters:
CICS_EpiInitializeExit	Version Anchor

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

This exit enables the user to clean up the EPI exit termination environment. Any storage acquired by CICS_EpiInitializeExit must be released in this exit.

Function name:	**Parameters:**
CICS_EpiTerminateExit	Anchor

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

To allow the user to select a server, or override the one passed to **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** in the **System** parameter.

Function name:	Parameters:
CICS_EpiAddTerminalExit	Anchor NameSpace System NetName DevType

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.

Note:

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 can safely select the system.

If however the calling application specifies a system name, then it might 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 can do so, but bear in mind the following.

- 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, so the exit routine can 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

This exit enables the user to find out the terminal ID allocated to a new EPI terminal, after a successful EPI call to **CICS_EpiAddTerminal**.

Function name:	Parameters:
CICS_EpiTermIdExit	Anchor TermIndex System

Purpose

CICS_EpiTermIdExit is provided for compatibility with older applications only. All new applications that use the EPI exits 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

This exit enables the user to retrieve information about the current EPI terminal.

Function name:	**Parameters:**
CICS_EpiTermIdInfoExit	Anchor Version TermIndex EpiDetails

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

This exit enables the user to see when an EPI transaction is started, and is used for information gathering purposes. This exit does not select a system, and does not return data.

Function name:	**Parameters:**
CICS_EpiStartTranExtendedExit	Anchor TermIndex TransId Data Size

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

This exit enables the user to see when an EPI transaction is started, and is used for information gathering purposes. This exit does not select a system, and does not return data.

Function name:	**Parameters:**
CICS_EpiStartTranExit	Anchor TransId Data Size

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

This exit enables the user to see when a reply is sent to an EPI transaction, and is used for information gathering purposes.

Function name:	**Parameters:**
CICS_EpiReplyExit	Anchor TermIndex Data Size

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

This exit enables the user to clean up EPI terminal data structures.

Function Name:	**Parameters:**
CICS_EpiDelTerminalExit	Anchor TermIndex

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

This exit enables the user to collect EPI event data.

Function name:	**Parameters:**
CICS_EpiGetEventExit	Anchor TermIndex Wait Event

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

This exit enables the user to supply a new EPI system ID, if the value supplied by **CICS_Epi_AddTerminal** or **CICS_EpiAddExTerminal** is not valid.

Function name:	Parameters:
CICS_EpiSystemIdExit	Anchor NameSpace System NetName DevType FailedSystem Reason SubReason UserId PassWord

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 187 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

This exit enables the user to collect data if an EIP transaction ends abnormally or if an EPI terminal fails.

Function Name:	**Parameters:** Anchor
CICS_EpiTranFailedExit	TermIndex
	Wait
	Event

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 might not even occur on the same event as the CICS_EPI_EVENT_END_TRAN. If the exit requires to handle this situation, it can 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. Code pages

This table provides the following information about each code page: canonical name, description and CCSID.

A canonical name is converted to the corresponding CCSID so that a CICS server can determine the code page location for a data stream. The CICS server must support EPI Version 2 for the encodings to be implemented. To find out which CCSIDs your CICS server supports check your CICS Server documentation.

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
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

Canonical name	Description	CCSID
Cp871	IBM Iceland	871
Cp874	IBM Thai	9066
Cp875	IBM Greek	875
Cp918	IBM Pakistan (Urdu)	918
Cp921	IBM Latvia, Lithuania (IBM AIX®, DOS)	921
Cp922	IBM Estonia (IBM 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	IBM AIX Chinese (Taiwan)	964
Cp970	IBM AIX Korean	970
Cp1006	IBM AIX Pakistan (Urdu)	1006
Cp1025	IBM Multilingual Cyrillic: Bulgaria, Bosnia, Herzegovinia, 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

Canonical name	Description	CCSID
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
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

Part 2. Appendixes

Glossary

This glossary defines the terms and abbreviations used in CICS Transaction Gateway and in the information centers.

A

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 systems.

APAR See *Authorized program analysis report*.

API See *application programming interface*.

APPC See *Advanced program-to-program communication*.

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

1. On CICS Transaction Gateway: The application identifier that is used to identify connections on the CICS server and tasks in a CICSplex. See also *APPLID qualifier* and *fully qualified APPLID*.
2. On CICS Transaction Server: 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.

APPLID qualifier
Optionally used as a high-level qualifier for the APPLID to form a fully qualified APPLID. See also *APPLID* and *fully qualified APPLID*.

ARM See *automatic restart manager*.

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.

Authorized Program Analysis Report (APAR)
A request for correction of a defect in a current release of an IBM-supplied program.

autoinstall
A method of creating and installing resources dynamically as terminals log on, and deleting them at logoff.

automatic restart manager (ARM)
An IBM 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.

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.

B

Basic Mapping Support
Basic mapping support is an interface between CICS and CICS application programs that move 3270 data streams to and from a terminal. The format of the input and output display data is defined by the BMS commands.

bean A definition or instance of a JavaBeans component. See also *JavaBeans*.

bean-managed transaction
A transaction where the JEE 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).

BMS see Basic Mapping Support

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*.

C

CA See *certificate authority*.

callback
A way for one thread to notify another application thread that an event has happened.

CCIN The CCIN transaction is invoked by the Client daemon, for each TCP/IP or SNA connection established. CCIN installs a Client connection on the CICS server.

CCSID
Coded Character Set Identifier. A 16-bit number that includes a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and other information that uniquely identifies the coded graphic-character representation.

certificate authority (CA)
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
The Client daemon, the EXCI (External CICS Interface), and the IPIC (IP Interconnectivity) protocol are collectively called the 'CICS connectivity components'. The Client daemon handles the TCP/IP and the SNA protocols.

CICS Request Exit
An exit that is invoked by the CICS Transaction Gateway for IBM z/OS at run time to determine which CICS server to use.

CICS server name
A defined server known to CICS Transaction Gateway.

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 interact with 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 that uses one or more of the CICS Transaction Gateways APIs.

Client daemon
The Client daemon manages TCP/IP and SNA connections to CICS servers on UNIX, Linux, and Windows. It processes ECI, EPI, and ESI requests, sending and receiving the appropriate flows to and from the CICS server to satisfy Client application requests. It can support concurrent requests to one or more CICS servers. The CICS Transaction Gateway initialization file defines the operation of the Client daemon and the servers and protocols used for communication.

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.

COMMAREA
See *communication area*.

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.

CTIN The CTIN transaction is invoked by the Client daemon to install a Client terminal definition on the CICS server.

D

daemon
A program that runs unattended to perform continuous or periodic systemwide functions, such as network control. A daemon can 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*.

default CICS server
The CICS server that is used if a server name is not specified on an ECI, EPI, or ESI request. The default CICS server name is defined as a product wide setting in the configuration file (ctg.ini).

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.

distinguished name
The name that uniquely identifies an entry in a directory. A distinguished name is made up of attribute:value pairs, separated by commas. The format of a distinguished name is defined by RFC4514. For more information, see `http://www.ietf.org/rfc/rfc4514.txt`. See also *realm name* and *identity propagation*.

distributed application
An application for which the component application programs are distributed between two or more interconnected processors.

distributed identity
User identity information that originates from a remote system. The distributed identity is created in one system and is passed to one or more other systems over a network. See also *distinguished name* and *realm name*.

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.

DLC See *data link control*.

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 IP 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.

dynamic server selection (DSS)
The mapping of a logical CICS server name to an actual CICS server name at run time.

E

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 JEE applications. There are three types of enterprise beans: entity beans, session beans, and message-driven beans.

Enterprise Information System (EIS)
The applications that comprise an enterprise's existing system for handling company-wide information. An enterprise information system offers a well-defined set of services that are exposed as local or remote interfaces or both.

Enterprise JavaBeans (EJB)
A component architecture defined by Oracle for the development and deployment of object-oriented, distributed, enterprise-level applications (JEE).

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*.

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 call interface (ECI)
A facility that allows a non CICS program to run a CICS program. Data is exchanged in a COMMAREA or a channel as for usual CICS interprogram communication.

external communications interface (EXCI)
An MVS™ application programming interface provided by CICS Transaction Server for IBM z/OS that enables a non-CICS program to call a CICS program and to pass and receive data using a COMMAREA. The CICS application program is started 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.

External Security Manager (ESM)
A security manager that operates outside CICS. For example, RACF® can be used as an external security manager with CICS Transaction Server.

F

firewall
A configuration of software that prevents unauthorized traffic between a trusted network and an untrusted network.

FMH See *function management header*.

fully qualified APPLID
Used to identify CICS Transaction Gateway connections on the CICS server and tasks in a CICSplex. It is composed of an APPLID with an optional network qualifier. See also *APPLID* and *APPLID qualifier*.

function management header (FMH)
One or more headers, optionally present in the leading request units (RUs) of an RU chain, that allow one LU to (a) select a transaction program or device at the session partner and control the way in which the user data it sends is handled at the destination, (b) change the destination or the characteristics of the data during the session, and (c) transmit between session partners status or user information about the destination (for example, a program or device). Function management headers can be used with LU type 1, 4, and 6.2 protocols.

G

Gateway
A device or program used to connect two systems or networks.

Gateway classes
The Gateway classes provide APIs for ECI, EPI, and ESI that allow communication between Java client applications and the Gateway daemon.

Gateway daemon
A long-running Java process that listens for network requests from remote Client applications. It issues these requests to CICS servers using the CICS connectivity components. The Gateway daemon on IBM z/OS processes ECI requests and on UNIX, Windows, and Linux platforms it process EPI and ESI requests as well. The Gateway daemon uses the GATEWAY section of ctg.ini for its configuration.

Gateway group
A set of Gateway daemons that share an APPLID qualifier, and where each Gateway daemon has a unique APPLID within the Gateway group.

Gateway token
A token that represents a specific Gateway daemon, when a connection is established successfully. Gateway tokens are used in the C language statistics and ECI V2 APIs.

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.

H

HA group
See *highly available Gateway group*.

highly available Gateway group (HA group)
A Gateway group that utilizes TCP/IP load balancing, and can be viewed

as a single logical Gateway daemon. A Gateway daemon instance in a HA group can recover indoubt XA transactions on behalf of another Gateway daemon within the HA group.

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 IP 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 can 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 (HTTP)
In the Internet suite of protocols, the protocol that is used to transfer and display hypertext and XML documents.

Hypertext Transfer Protocol Secure (HTTPS)
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.

I

ID data
An ID data structure holds an individual result from a statistical API function.

identity propagation
The concept of preserving a user's security identity information (the distributed identity) independent of where the identity information has been created, for use during authorization and for auditing purposes. The distributed identity is carried with a request from the distributed client application to the CICS server, and is incorporated in the access control of the server as part of the authorization process, for example, using RACF. CICS Transaction Gateway flows the distributed identity to CICS. See also *distributed identity*.

identity propagation login module
A code component that provides support for identity propagation. The identity propagation login module is included with the CICS Transaction

Gateway ECI resource adapter (cicseci.rar), conforms to the JAAS specification and is contained in a single Java class within the resource adapter. See also *identity propagation*.

iKeyman
A tool for maintaining digital certificates for JSSE.

in doubt
The state of a transaction that has completed the prepare phase of the two-phase commit process and is waiting to be completed.

in flight
The state of a transaction that has not yet completed the prepare phase of the two-phase commit process.

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*.

<install_path>
This term is used in file paths to represent the directory where you installed the product. For more information, see ../installing/topics/cclahlinstfiles.dita.

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, run 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.

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.

IPIC See *IP interconnectivity*.

J

Java An object-oriented programming language for portable interpretive code that supports interaction among remote objects.

Java 2 Platform, Enterprise Edition (J2EE, Java EE)
An environment for developing and deploying enterprise applications, defined by Oracle. The JEE platform consists of a set of services, application programming interfaces (APIs), and protocols that allow multi-tiered, Web-based applications to be developed.

JavaBeans
As defined for Java by Oracle, 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 Oracle provided for the Java platform.

JavaGateway
The URL of the CICS Transaction Gateway with which the Java Client application communicates. The JavaGateway takes the form `protocol://address:port`. These protocols are supported: `tcp://`, `ssl://`, and `local:`. 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).

JavaScript Object Notation (JSON)
A lightweight data-interchange format that is based on the object-literal notation of JavaScript. JSON is programming-language neutral but uses conventions from languages that include C, C++, C#, Java, JavaScript, Perl, Python.

JCA See *JEE Connector Architecture*.

JDK See *Java development kit*.

JEE (formerly J2EE)
See *Java 2 Platform Enterprise Edition*.

JEE Connector architecture (JCA)
A standard architecture for connecting the JEE platform to heterogeneous enterprise information systems (EIS).

JNI See *Java Native Interface*.

JRE See *Java Runtime Environment*.

JSON See *JavaScript Object Notation (JSON)*.

JSON Schema
A JavaScript Object Notation (JSON) document that describes the structure and constrains the contents of other JSON documents.

JSON web service
A web service that accepts and produces JSON payloads.

JSSE See *Java Secure Socket Extension*.

JVM See *Java Virtual Machine*.

K

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.

Keystore
In the JSSE protocol, a file that contains public keys, private keys, trusted roots, and certificates.

L

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 CICS server
An alias that can be passed on an ECI request when running in remote mode to CICS Transaction Gateway. The alias name is mapped to an actual CICS server name by a dynamic server selection (DSS) mechanism.

logical end of day
The local time of day on the 24-hour clock to which a Gateway daemon aligns statistics intervals. If the statistics interval is 24 hours, this is the local time at which interval statistics will be reset and, on IBM z/OS, optionally recorded to SMF. This time is set using the **stateod** parameter in the configuration file (ctg.ini).

logical unit (LU)
In SNA, a port through which a user accesses the SNA network to communicate with another user and through which the 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 might be capable of supporting many sessions with other logical units. See also *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)
The processing that a program performs between synchronization points.

LU See *logical unit*.

LU 6.2 See *logical unit 6.2*.

LU-LU session
In SNA, a session between two logical units (LUs) in an SNA network. It provides communication between two users, or between a 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*.

M

managed mode
Describes an environment in which connections are obtained from connection factories that the JEE server has set up. Such connections are owned by the JEE server.

media 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.

N

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 IP addresses.

NAU See *network addressable unit*.

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 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.

nonextended logical unit of work
See *SYNCONRETURN*.

nonmanaged mode
An environment in which the application is responsible for generating and configuring connection factories. The JEE server does not own or know about these connection factories and therefore provides no Quality of Service facilities.

O

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.

OO See *object-oriented*.

OSGi A specification that describes a modular system and a service platform for the Java programming language that implements a complete and dynamic component model.

P

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.

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.

password phrase
A character string, between 9 and 100 characters in length, that is used for authentication when a user signs on to CICS. Because a password phrase can provide an exponentially greater number of possible combinations of characters than a standard 8 character password, the use of password

phrases can enhance system security. Password phrases are verified by the External Security Manager (ESM), and can contain alphanumeric characters, and any of the other non alphanumeric characters that are supported by the ESM. See also *External Security Manager (ESM)*.

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.

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.

port sharing
A way of load balancing TCP/IP connections across a group of servers running in the same IBM z/OS image.

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*.

<product_data_path>
This term represents the directory used by the Windows CICS Transaction Gateway for common application data. For more information, see ../installing/topics/cclahlinstfiles.dita.

protocol boundary
The signals and rules governing interactions between two components within a node.

Q

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.

R

RACF See *Resource Access Control Facility*.

realm A named collection of users and groups that can be used in a specific security context. See also *distinguished name* and *identity propagation*.

Recoverable resource management services (RRMS)
The registration services, context services, and resource recovery services provided by the IBM z/OS sync point manager that enable consistent changes to be made to multiple protected resources.

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 monitoring exits
Exits that provide information about individual requests as they are processed by the CICS Transaction Gateway.

request unit (RU)
In SNA, a message unit that contains control information such as a request code, or function management (FM) headers, 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 also *CID methodology*.

response unit (RU)
A message unit that acknowledges a request unit; it can contain prefix information received in a request unit.

Resource adapter
A system-level software driver that is used by an EJB container or an application client to connect to an enterprise information system (EIS). A resource adapter plugs in to a container; the application components deployed on the container then use the client API (exposed by adapter) or tool-generated, high-level abstractions to access the underlying EIS.

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)
An IBM z/OS facility that provides two-phase sync point support across participating resource managers.

RESTful
Pertaining to applications and services that conform to Representational State Transfer (REST) constraints.

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.

RPC See *remote procedure call*.

RRMS
See *Recoverable resource management services*.

RRS See *Resource Recovery Services*.

RU See *Request unit* and *Response unit*.

S

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.

server name remapping
See *dynamic server selection*.

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.

sign-on capable terminal
A sign-on capable terminal allows sign-on transactions that are either supplied with CICS (CESN) or written by the user, to be run. Contrast with sign-on incapable terminal.

silent installation
Installation that does not display messages or windows during its progress. Silent installation is not a synonym of "unattended installation", although it is often improperly used as such.

single-byte character set (SBCS)
A character set in which each character is represented by 1 byte. Contrast with double-byte character set.

SIT See *system initialization table*.

SLU See *secondary logical unit*.

SMF The IBM z/OS System Management Facility (SMF) collects and records system and job-related information that your IBM z/OS installation can use for reporting, billing, analysis, profiling, and maintaining system security. CICS TG for IBM z/OS writes statistical data to SMF.

SMIT See *System Management Interface Tool*.

SNA See *Systems Network Architecture*.

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*.

SSLight
An implementation of SSL, written in Java, and no longer supported by CICS Transaction Gateway.

standard error
In many workstation-based operating systems, the output stream to which error messages or diagnostic messages are sent.

statistic data
A statistic data structure holds individual statistical result returned after calling a statistical API function.

statistic group
A generic term for a collection of statistic IDs.

statistic ID
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.

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
Synchronization point. During transaction processing, a reference point to which protected resources can be restored if a failure occurs.

SYNCONRETURN
A request where the CICS server takes a sync point on successful completion of the server program. Changes to recoverable resources made by the server program are committed or rolled-back independently of changes to recoverable resources made by the client program issuing the

ECI request, or changes made by the server in any subsequent ECI request. Also referred to as a *nonextended logical unit of work*.

system initialization table (SIT)
A table containing parameters used to start a CICS control region.

System Management Command
An administrative request received by a Gateway daemon (or Gateway daemon address space on IBM z/OS) from the **ctgadmin** command (on UNIX, Linux, or Windows) or the IBM z/OS console. The request might be made to retrieve information about the Gateway daemon, or to alter some aspect of Gateway daemon behavior. Typically, a **ctgadmin** command in the form **ctgadmin** <command string> is entered by an operator using the command line interface, or a modify command in the form /F <job name>,APPL=<command string> is entered by an operator on the IBM z/OS console.

System Management Interface Tool (SMIT)
An interface tool of the IBM AIX operating system for installing, maintaining, configuring, and diagnosing tasks.

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 IBM z/OS.

T

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 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*.

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 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.

U

unattended installation
Unattended installation is installation performed without user interaction during its progress, or, with no user present at all, except for the initial launch of the process.

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.

UOW See *unit of work*.

UR See *unit of recovery*.

URL See *Uniform Resource Locator*.

user registry
The location where the distinguished name of a user is defined and authenticated. See also *distinguished name*.

user session
Any APPC session other than a SNASVCMG session.

V

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.

W

WAN See *wide area network*.

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 can use or provide public communication facilities.

Wrapping trace
On Windows, UNIX, and Linux, 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**.

WSBind file
A Web service bind file is a resource that describes the specifics of the Web service.

X

XA request
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).

Related literature

Other documentation relating to CICS Transaction Gateway.

IBM Redbooks® titles are available on a wide range of subjects relevant to CICS Transaction Gateway programming, installation, operation and troubleshooting. See the: IBM Redbooks site for more information.

Documentation for many IBM products is available online from the IBM Publications Center.

Accessibility

Accessibility features help users with a physical disability, for example restricted mobility or limited vision, to use information technology products successfully. CICS Transaction Gateway is compatible with the JAWS screen reader. CICS Transaction Gateway provides accessibility by enabling keyboard-only operation.

For more information about the IBM commitment to accessibility, visit the IBM Accessibility Center.

Installation

The InstallAnywhere wizard is not fully accessible to screen readers.

To use the installer with a screen reader you must use console mode installation from a command prompt, specifying the -i console option.

Console mode displays text over multiple screens, and enables you to make choices during the installation process. The command prompt interface does not provide a cursor for navigating over the displayed text. When you use the JAWS screen reader you can repeat the displayed text with the command used for reading the current window (key combination Insert+B).

The first screen displayed by the installer is for language selection; the default language depends on the values in the system regional settings. To bypass the language selection screen, use the -l *lang* command option; where *lang* is one of the following:

- `de` German
- `en` English
- `es` Spanish
- `fr` French
- `it` Italian
- `ja` Japanese
- `ko` Korean
- `tr` Turkish
- `zh_CN` Chinese

For example, to install with the console interface in French:

```
installer -i console -l fr
```

Configuration Tool accessibility

The configuration file uses the number sign (#) character to denote a comment; consider configuring your screen reader accordingly.

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'.
```

When using a screen reader on Windows, the Gateway daemon should be started and stopped with Windows services by starting and stopping the IBM CICS Transaction Gateway service. To determine if the CICS Transaction Gateway has started or stopped use the Windows Event Log viewer to check the messages in the Application log.

cicsterm

Although cicsterm is accessible, it relies on the application that is being processed to define an accessible 3270 screen.

Keyboard mapping depends on the terminal type that you are using, for more information, see *Keyboard mapping for cicsterm*.

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.

Uppercase
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.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property 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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

The client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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-IBMproducts. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the

names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Programming interface information

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

Safety and environmental notices

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Readers' Comments — We'd Like to Hear from You

CICS Transaction Gateway for Multiplatforms
Version 9 Release 2
Programming Reference

Publication No. SC34-7340-00

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:
- Send your comments to the address on the reverse side of this form.
- Send a fax to the following number: +44 1962 816151
- Send your comments via email to: idrcf@uk.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC34-7340-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM United Kingdom Limited
User Technologies Department (MP189)
Hursley Park
Winchester
Hampshire
United Kingdom
SO21 2JN

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

SC34-7340-00