

CICS® Family



Client/Server Programming

CICS® Family



Client/Server Programming

Note!

Before using this information and the product it supports, read the information under “Appendix D. Notices” on page 207.

Fifth edition (September 1999)

This edition replaces SC33-1435-03.

© **Copyright International Business Machines Corporation 1989, 1999. All rights reserved.**

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

Contents

Figures	vii	ECI_STATE_SYNC call type	35
Tables	ix	ECI_STATE_ASYNC call type	39
Summary of changes	xi	ECI_GET_REPLY call type	44
Changes to the fifth edition	xi	ECI_GET_REPLY_WAIT call type	48
About this book	xiii	ECI_GET_SPECIFIC_REPLY call type	49
Who should read this book.	xiii	ECI_GET_SPECIFIC_REPLY_WAIT call type.	53
Conventions and terminology used in this book	xiii	ECI status block.	54
Prerequisite and related information	xiii	CICS_EciListSystems	55
General	xiii	Chapter 3. External presentation interface 57	
Setting up client-server systems	xiv	Overview	57
Client application programming	xiv	How to use the EPI	58
Application programming on CICS servers	xiv	Initialization and termination	58
Miscellaneous	xiv	Listing the configured servers	59
How to send your comments	xv	Adding terminal resources	59
Obtaining books from IBM.	xvi	Deleting terminal resources	61
Chapter 1. Introducing the external access interfaces	1	Authentication and authorization	61
Overview	1	Starting transactions	62
External call interface	3	Events and callbacks	63
External presentation interface	4	Processing the events	64
External security interface.	5	Sending and receiving data	65
Using the external access interfaces	6	Managing pseudoconversations	65
ECI and EPI exits	6	Security in the EPI	66
Chapter 2. External call interface.	7	EPI constants and data structures	67
Overview	7	EPI constants	68
ECI function	7	EPI data structures.	68
Types of ECI calls	8	EPI versions	77
Program link calls	9	EPI functions.	77
Managing logical units of work	9	CICS_EpiInitialize	82
Security in the ECI.	12	CICS_EpiTerminate	83
Status information calls	13	CICS_EpiListSystems	84
How status information is supplied and used	13	CICS_EpiAddTerminal	86
Reply solicitation calls	15	CICS_EpiAddExTerminal.	91
CICS_ExternalCall	16	CICS_EpiInquireSystem	96
ECI_SYNC call type	19	CICS_EpiDelTerminal	97
ECI_ASYNC call type	27	CICS_EpiPurgeTerminal	98
		CICS_EpiSetSecurity	99
		CICS_EpiStartTran	101
		CICS_EpiReply	104
		CICS_EpiATISate.	106
		CICS_EpiSenseCode	108
		CICS_EpiGetEvent	110
		CICS_EpiGetSysError	112
		EPI events	114
		CICS_EPI_ADD_TERM	115

CICS_EPI_EVENT_SEND	116
CICS_EPI_EVENT_CONVERSE	117
CICS_EPI_EVENT_END_TRAN	118
CICS_EPI_EVENT_START_ATI	119
CICS_EPI_EVENT_END_TERM	120
3270 data streams for the EPI	121
Inbound data streams (EPI to CICS).	121
Outbound data streams (CICS to EPI)	122
3270 order codes	123
Microsoft Windows 3.1 considerations	125
Chapter 4. Creating ECI and EPI application programs	127
Writing the non-CICS applications	127
Making ECI calls	128
CICS_ExternalCall	129
Callback routines	130
CICS_EciListSystems	130
Debugging with REXX	131
Making EPI calls	132
EPI functions	132
Callback routines	133
Compiling and linking applications	133
IBM CICS Client for DOS Version 2	133
IBM CICS Client for Windows Version 2	134
IBM CICS Clients for Windows NT and Windows 95 Version 2	134
IBM CICS Client for OS/2 Version 2	135
IBM CICS Universal Clients for Windows NT and Windows 98 Version 3	138
IBM CICS Universal Client for OS/2 Version 3	139
IBM CICS Universal Client for AIX Version 3	140
IBM CICS Universal Client for Solaris Version 3	140
CICS for OS/2 Version 3 server implementation	141
Chapter 5. External security interface	143
Overview	143
Benefits of APPC PEM	144
Benefits of the ESI	144
ESI constants and data structures	144
ESI constants	145
ESI data structures	145
ESI functions	149
CICS_VerifyPassword	150
CICS_ChangePassword	153
CICS_SetDefaultSecurity	156

Appendix A. ECI extensions that are environment-dependent	159
Call type extensions	159
Asynchronous program link call, with notification by message (ECI_ASYNC_NOTIFY_MSG)	159
Asynchronous program link call, with notification by semaphore (ECI_ASYNC_NOTIFY_SEM)	160
Asynchronous status call, with notification by message (ECI_STATE_ASYNC_MSG)	160
Asynchronous status call, with notification by semaphore (ECI_STATE_ASYNC_SEM)	161
Time-outs	161
Fields to support ECI extensions	162
Reply message formats	164
ECI return notification	164
Summary of input parameter requirements	165

Appendix B. CICS Universal Client Programming Samples	167
--	------------

Appendix C. ECI and EPI exits	169
Installing the exits	169
Exit routine environment	170
How the exit routines are described in the reference sections	171
ECI exits reference	171
Identification token	172
EPI exits reference	184
CICS_EpiInitializeExit	186
CICS_EpiTerminateExit	187
CICS_EpiAddTerminalExit	188
CICS_EpiTermIdExit	191
CICS_EpiTermIdInfoExit	192
CICS_EpiStartTranExit	193
CICS_EpiReplyExit	194
CICS_EpiDelTerminalExit	195
CICS_EpiGetEventExit	196
CICS_EpiSystemIdExit	197
CICS_EpiTranFailedExit	199
Diagnostic information	201
CICSTERM, CICSPRNT and the EPI exits	201

Appendix D. Notices	207
Programming interface information	209
Trademarks	209

Index. 211

Figures

1. External access interfaces in CICS client/server configurations	2	4. External presentation interface	4
2. Server implementation of the external interfaces.	2	5. Asynchronous program link calls with message qualifiers and LUW tokens . . .	12
3. External call interface illustrated	3	6. Using terminal indexes.	60

Tables

1. Logical units of work in the ECI . . . 10
2. Status enquiries with CICS_ExternalCall 14
3. Summary of EPI functions 78
4. Order codes occurring in 3270 data streams 123
5. CICS_ExternalCall return codes — environment-dependent extensions . . 164
6. Input parameters for CICS_ExternalCall — environment-dependent extensions . 166
7. ECI and EPI exits 169
8. Summary of ECI exits. 171
9. Summary of EPI exits. 184

Summary of changes

The following are the changes since the previous edition of this book.

Changes to the fifth edition

Information has been added about:

- Adding and deleting terminal resources, (in the External presentation interface chapter).
- The **CICS_EpiAddExTerminal** function.
- The **CICS_EpiPurgeTerminal** function.
- The **CICS_EpiSetSecurity** function.
- The **CICS_EPI_EVENT_ADD_TERM** event.
- The **CICS_EpiAttributes_t** structure.
- The **System TermId** and **SignonCapability** fields in the **CICS_EpiDetails_t** structure.
- The **MapName** and **MapSetName** fields in the **CICS_EpiEventData_t** structure.
- The **CICS_SetDefaultSecurity** function.
- The ECI exit **CICS_EciSetProgramAliasExit**.
- The EPI exit **CICS_EpiTermidInfoExit**.
- The programming samples supplied with CICS Universal Clients.

Information about CICS on Open Systems has been removed.

Various minor technical corrections have been made.

About this book

This book is about the CICS Family programming interfaces to enable non-CICS applications to use CICS facilities in a client-server environment.

The interfaces described are:

- External call interface (ECI)
- External presentation interface (EPI).
- External security interface (ESI).

Who should read this book

This book is for application designers and application programmers in a client-server environment.

You should have a good knowledge of:

- CICS, and the CICS servers that the applications will use
- The concepts of client/server programming
- The programming language in which the applications will be written
- The programming environment in which the programs will operate.

Conventions and terminology used in this book

The presentation of names of program elements is as follows:

1. When a program element is in lowercase, or mixed case, it is always written in this book in a bold face: **eci_userid**, **CICS_EpiAddTerminal**.
2. When a program element is in uppercase, it is always written in this book in a normal face: ECI_ERR_LUW_TOKEN, CICS_EPI_EVENT_END_TERM.

“CICS on System/390[®]” is used to refer to:

- CICS/VSE[®]
- CICS/MVS[®]
- CICS/ESA[®]
- CICS Transaction Server for OS/390[®]

Prerequisite and related information

General

- *CICS Family: Interproduct Communication*, SC33-0824

Setting up client-server systems

- *CICS Clients Administration*
 - SC33-1792 for CICS Clients Version 2
 - SC34-5450 for CICS Universal Client for OS/2[®] Version 3
 - SC34-5449 for CICS Universal Client for Windows[™] Version 3
 - SC34-5448 for CICS Universal Client for AIX[®] Version 3
 - SC34-5451 for CICS Universal Client for Solaris Version 3
- *CICS for OS/2 Customization*, SC33-1581
- *CICS for OS/2 Intercommunication*, SC33-1583
- *IBM Transaction Server for Windows NT[™] Version 4 Quick Beginnings*, GC33-1879
- *IBM Transaction Server for Windows NT Version 4 Administration Guide*, GC33-1881
- *IBM Transaction Server for Windows NT Version 4 Administration Reference (CICS)*, GC33-1885
- *CICS/400[®] Administration and Operations Guide*, SC33-1387
- *CICS/400 Intercommunication*, SC33-1388
- *CICS Transaction Server for OS/390 Resource Definition Guide*, SC33-1684
- *CICS Transaction Server for OS/390 Customization Guide*, SC33-1683
- *CICS/VSE Resource Definition*, SC33-0708
- *CICS/VSE Customization Guide*, SC33-0707

You may need to consult other publications relevant to your client and server systems.

Client application programming

For information about application programming on client systems, please consult the publications for the client systems.

Application programming on CICS servers

Application programming on CICS servers is described in publications in the library for each server environment.

Miscellaneous

- *An Introduction to the IBM 3270 Information Display System*, GA27-2739
- *IBM 3270 Information Display System Data Stream Programmer's Reference*, GA23-0059
- *Guide to Writing DCE Applications*, by John Shirley, published by O'Reilly Associates, Sebastopol, CA, USA, ISBN 1-56592-004-X

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book, or any other CICS documentation:

- Visit our Web site at:

<http://www.ibm.com/software/ts/cics/>

and follow the **library** link to our feedback form.

Here you will find the feedback page where you can enter and submit your comments.

- Send your comments by e-mail to idrctf@hursley.ibm.com
- Fax your comments to:

+44-1962-870229 (if you are outside the UK)
01962-870229 (if you are in the UK)

- Mail your comments to:

Information Development
Mail Point 095
IBM United Kingdom Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN
United Kingdom

Whichever method you use, ensure that you include:

- The name of the book
- The form number of the book
- If applicable, the version of the product
- The specific location of the text you are commenting on, for example, a page number or table number.

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Obtaining books from IBM

For information on books you can download, visit our Web site at:

<http://www.ibm.com/software/ts/cics/>

and follow the **library** link.

You can order hardcopy books:

- Through your IBM representative or the IBM branch office serving your locality.
- By calling 1-800-879-2755 in the United States.
- From the Web site at:

<http://www.elink.ibm.link.ibm.com/pb1/pb1>

Chapter 1. Introducing the external access interfaces

This chapter introduces CICS Family Client/Server Programming, which comprises two application programming interfaces (APIs) that provide external access to CICS facilities:

- External call interface (ECI)
- External presentation interface (EPI)
- External security interface (ESI)

The chapter is organized as follows:

“Overview”

“External call interface” on page 3

“External presentation interface” on page 4

“External security interface” on page 5

“Using the external access interfaces” on page 6

“ECI and EPI exits” on page 6

Overview

The ECI and EPI allow your non-CICS applications to gain access to CICS facilities and data.

Figure 1 on page 2 illustrates the use of the external interfaces by a non-CICS application in a client system. This application is using the facilities of CICS in a server system. The CICS client software processes the application’s ECI and EPI requests, and transmits them to the server system using an appropriate communication protocol. Although the figure shows the client system and server system as separate workstations, it is possible for the whole configuration to be on a single workstation.

Introduction

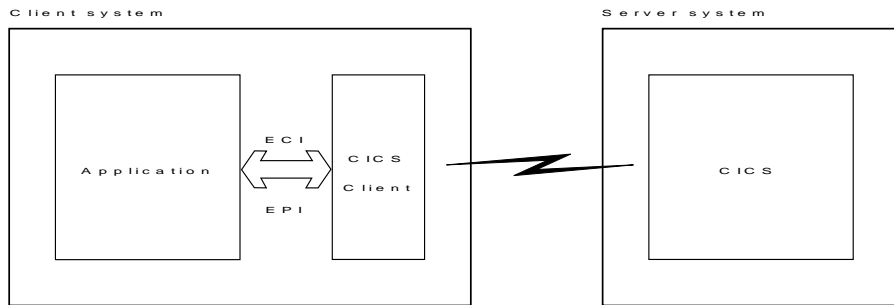


Figure 1. External access interfaces in CICS client/server configurations

Some members of the CICS family provide the external interfaces to non-CICS applications without the use of a CICS client. The non-CICS application *must be on the same workstation as the server*, and is not able to communicate with other servers. This is illustrated in Figure 2. In this case the application is using the server implementation of the external interfaces.

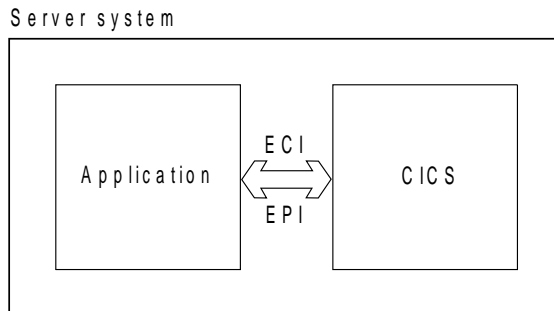


Figure 2. Server implementation of the external interfaces

The following can be client systems that can connect to any CICS server. In these systems an appropriate CICS client must be installed. See the appropriate *CICS Clients Administration* manual.

- OS/2 — IBM CICS Client for OS/2 Version 2 and IBM CICS Universal Client for OS/2 Version 3
- DOS — IBM CICS Client for DOS Version 2
- Microsoft™ Windows — IBM CICS Client for Windows Version 2
- Microsoft Windows NT — IBM CICS Client for Windows NT Version 2 and IBM CICS Universal Client for Windows NT Version 3
- Microsoft Windows 95 — IBM CICS Client for Windows 95 Version 2
- IBM CICS Universal Client for Windows 98 Version 3
- IBM CICS Universal Client for AIX Version 3
- IBM CICS Universal Client for Solaris Version 3

The following server systems provide a server implementation of the external interfaces:

- IBM CICS for OS/2
- IBM Transaction Server for OS/2 Warp

Any member of the CICS family can be a server, though CICS on System/390 servers before CICS for MVS/ESA Version 4 Release 1 with PTF UN90142 support only the ECI.

External call interface

The ECI allows a non-CICS application to call a CICS program in a CICS server. The application can be connected to several servers at the same time, and it can have several program calls outstanding at the same time.

The CICS program cannot perform terminal I/O, but can access and update all other CICS resources.

Figure 3 shows that the same CICS program can be called by a non-CICS application using the external call interface, or by a CICS program using EXEC CICS LINK. Data is exchanged between the two programs by means of a COMMAREA, in a similar way to CICS interprogram communication.

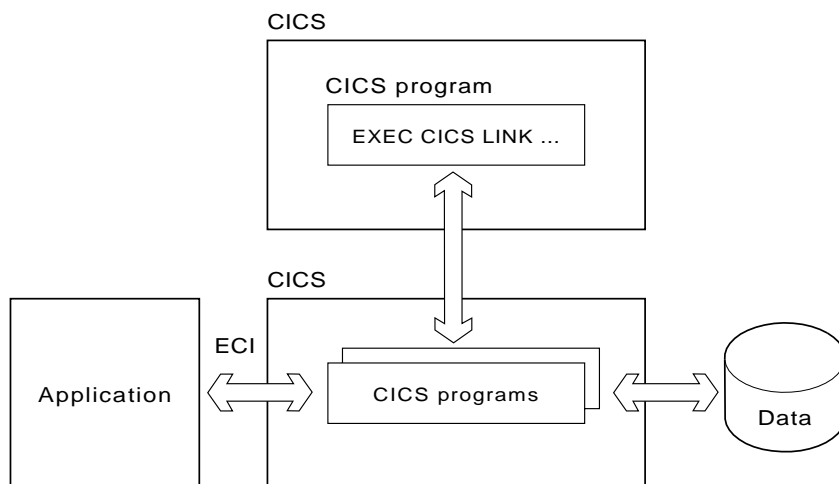


Figure 3. External call interface illustrated

Calls may be made synchronously or asynchronously. Synchronous calls return control when the called program completes, and the information returned is immediately available. Asynchronous calls return control without

Introduction

reference to the completion of the called program, and the application can ask to be notified when the information becomes available.

Calls may also be *extended*. That is, a single logical unit of work may cover two or more successive calls, though only one call can be active for each logical unit of work at any time. The application can manage many logical units of work concurrently if it uses asynchronous calls.

The called program can update resources on its own system, it can use distributed program link (DPL) to call CICS programs on other systems, and it can access resources on other CICS systems by function shipping, by distributed transaction processing (DTP), or (in the CICS/MVS and CICS/ESA environments) by the front end programming interface (FEPI).

External presentation interface

The EPI allows a non-CICS application program to be viewed as a 3270 terminal by a CICS server system to which it is connected. Figure 4 shows how both an EPI application and a CICS terminal can schedule transactions in a CICS server.

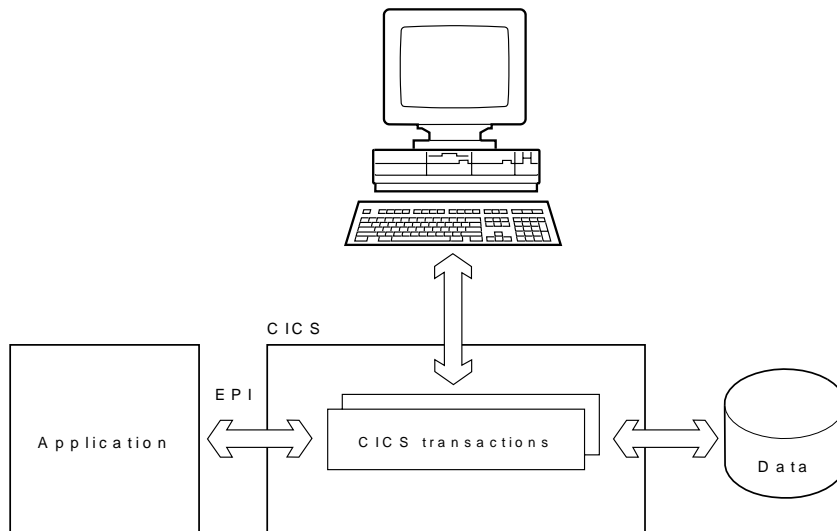


Figure 4. External presentation interface

The application can be using the facilities of several servers at the same time, and can act as if it were many different 3270 terminals.

The application can schedule CICS transactions, and for these transactions it is the principal facility.

With CICS servers that support access through the EPI, other CICS transactions running in the server can use the CICS START command to schedule transactions that use the non-CICS application as their initiating terminal.

When a non-CICS application uses the EPI to start a transaction in a CICS server, 3270 data streams and events are passed between the server and the application. The application can present the contents of the terminal I/O to its user in any manner appropriate to the application's operating environment.

Transactions can be routed to other CICS systems by standard transaction routing. Resources on other CICS systems can be accessed by function shipping.

Note that server transactions can be existing transactions that use 3270 input and output (with some restrictions).

External security interface

The ESI allows a non-CICS application to invoke services provided by advanced program-to-program communication (APPC) password expiration management (PEM).

APPC PEM with CICS provides support for an APPC architected sign-on transaction that signs on userids to a CICS server, and processes requests for a password change by:

- Identifying a user and authenticating that user's identification
- Notifying specific users during the authentication that their passwords have expired
- Letting users change their passwords when (or before) the passwords expire
- Telling users how long their current passwords will remain valid
- Providing information about unauthorized attempts to access the server using a particular user identifier

ESI calls can be included within your ECI or EPI applications.

Introduction

Using the external access interfaces

The external interfaces allow non-CICS applications to access and update CICS resources by initiating CICS transactions or calling CICS programs. When used in conjunction with the CICS communication facilities, they enable non-CICS programs to access and update resources on any CICS system. This supports such activities as:

1. Developing graphical user interface (GUI) front ends for CICS applications, using Presentation Manager[®] or other presentation systems
2. Connecting external devices such as bar-code readers to CICS systems
3. Allowing the integration of CICS systems and non-CICS systems.

The EPI allows you to develop GUIs, either for existing CICS systems or for new applications. It is particularly useful for developing new GUI front ends for existing CICS transactions, which need not be changed. The application can use the EPI to communicate with a CICS transaction, and can exploit the presentation facilities of the client system to communicate with the end user.

If you attach an external device such as a bar code reader, the application can deal with device input and output, and can use the EPI to start a transaction, perhaps one already written to deal with the kind of data that the external device produces. The application converts the input from the external device into a 3270 data stream to start the transaction and pass the data to it. Output from the transaction, in the form of a 3270 data stream, is then converted into the signals and data streams that operate the external device.

The integration of CICS and non-CICS systems usually involves passing user-defined data between the programs of the non-CICS system and a CICS program, and the ECI can be used for this.

In any of these cases, the choice between EPI and ECI is not always clear-cut, because both interfaces can be used to pass data between a non-CICS application and a CICS program. However, the mechanism is different in the two cases: 3270 data streams for EPI; application-defined formats in a COMMAREA for ECI.

ECI and EPI exits

For CICS Clients Version 2, and CICS Universal Clients, the operation of the EPI and ECI can be customized by the user exits described in “Appendix C. ECI and EPI exits” on page 169.

Chapter 2. External call interface

This chapter provides reference information about the external call interface (ECI).

The interface is described here in a language-independent manner, though the names chosen for the elements of the interface—functions, parameters, data structures, fields, constants, and so on—are similar to those provided for programming. Language-dependent information is in “Chapter 4. Creating ECI and EPI application programs” on page 127.

On some platforms, the ECI provides facilities that are not part of CICS Family Client/Server Programming, and these are summarized in “Appendix A. ECI extensions that are environment-dependent” on page 159.

The chapter is organized as follows:

- “Overview”
- “Program link calls” on page 9
- “Status information calls” on page 13
- “Reply solicitation calls” on page 15
- “CICS_ExternalCall” on page 16
- “ECI status block” on page 54
- “CICS_EciListSystems” on page 55.

Overview

ECI function

The ECI allows a non-CICS application program to call a CICS program in a CICS server. The non-CICS application does not issue any CICS commands itself; the CICS commands are issued by the called program running in the server. The called program thus appears to have been called by EXEC CICS LINK with the COMMAREA option. An ECI application can make use of existing CICS programs that obey the rules for distributed program link, or new CICS programs can be written to be called by ECI applications. The called programs must follow the rules for CICS programs called by DPL.

External call interface

In some circumstances it is possible for the application to be running more than one CICS program concurrently, and these programs might be distributed across several CICS servers.

The function provided by the ECI is packaged in two parts as follows:

CICS_ExternalCall

Provides most of the function of the ECI. It has a single parameter, the ECI parameter block, in which various fields describe the function to be performed and the inputs and outputs. Most of the following sections are concerned with the use of **CICS_ExternalCall**.

CICS_EciListSystems

Used to find out about available servers to which **CICS_ExternalCall** requests can be directed. It is described in “CICS_EciListSystems” on page 55.

Types of ECI calls

The calls to **CICS_ExternalCall** are of three types:

- Program link calls
- Status information calls
- Reply solicitation calls.

The type of call to **CICS_ExternalCall** is controlled by the setting of the **eci_call_type** parameter in the ECI parameter block.

Restrictions on call type for particular operating environments are discussed in the descriptions of each call type.

Program link calls

Program link calls cause a CICS program to be executed on a CICS server. These calls can be:

- Synchronous — the application waits until the called program has finished. Returned information is immediately available.
- Asynchronous — the application gets control back without reference to the completion of the called program. The application can ask to be notified later when the information is available. It must use a reply solicitation call to determine the outcome of the asynchronous request.

Status information calls

Status information calls retrieve status information about the type of system on which the application is running and its status. These calls can be:

- Synchronous — the application waits until the information has been made available.
- Asynchronous — the application gets control back while the information is being retrieved. The application can ask to be notified later when the information is available. It must use a reply solicitation call to determine the outcome of the asynchronous request.

Reply solicitation calls

Reply solicitation calls get information back after asynchronous program link or asynchronous status information calls. Reply solicitation calls can be:

- General — retrieving any piece of outstanding information
- Specific — retrieving information for a named asynchronous request.

An application that uses the asynchronous method of calling may have several program link and status information calls outstanding at any time. The **eci_message_qualifier** parameter in the ECI parameter block can be used on an asynchronous call to provide a user-defined identifier for the call. The use of different identifiers for different asynchronous calls within a single application is the programmer's responsibility. When a general reply solicitation call is made, the ECI uses the **eci_message_qualifier** field to return the name of the call to which the reply belongs. When a specific reply solicitation call is made, you must supply a value in the **eci_message_qualifier** field to identify the asynchronous call about which information is being sought.

Program link calls

Program link calls can be either synchronous or asynchronous. For asynchronous calls, it is the responsibility of the calling application to solicit the reply using one of the reply solicitation calls. (See "Reply solicitation calls" on page 15.)

Managing logical units of work

An ECI application is often concerned with updating recoverable resources on the server, and the application programmer needs to understand the facilities that the ECI provides for managing logical units of work. A logical unit of work is all the processing in the server that is needed to establish a set of updates to recoverable resources. When the logical unit of work ends normally, the changes will all be committed. When the logical unit of work ends abnormally, for instance because a program abends, the changes are all backed out. You can use the ECI to start and end logical units of work on the server.

External call interface

The changes to recoverable resources in a logical unit of work might be effected by:

- A single program link call, or
- A sequence of program link calls.

On successful return from the first of a sequence of calls for a logical unit of work, the **eci_luw_token** field in the control block contains a token that should be used for all later calls related to the same logical unit of work. All program link calls for the same logical unit of work will be sent to the same server.

Attention: You should be careful when extending a logical unit of work across multiple program link calls that may span a long time (for example, over user think time). The reason is that the logical unit of work holds various locks and other CICS resources on the server, and this may cause delays to other users who are waiting for those same locks and resources.

When a logical unit of work ends, the CICS server attempts to commit the changes. The last, or only, program link call of a logical unit of work is advised whether the attempt was successful.

Only one program link call per logical unit of work can be outstanding at any time. (An asynchronous program link call is outstanding until a reply solicitation call has processed the reply.)

Table 1 shows how you can use combinations of **eci_extend_mode**, **eci_program_name**, and **eci_luw_token** parameter values to perform tasks associated with managing logical units of work through the ECI. In each case you must also store appropriate values in other fields for the call type you have chosen.

Table 1. Logical units of work in the ECI

Task to perform	Parameters to use
Call a program that is to be the only program of a logical unit of work.	Set up the parameters as follows: <ul style="list-style-type: none">• eci_extend_mode: ECI_NO_EXTEND• eci_program_name: provide it• eci_luw_token: zero
Call a program that is to start a new logical unit of work that is to be extended.	Set up the parameters as follows: <ul style="list-style-type: none">• eci_extend_mode: ECI_EXTENDED• eci_program_name: provide it• eci_luw_token: zero Afterwards, save the token from eci_luw_token .

Table 1. Logical units of work in the ECI (continued)

Task to perform	Parameters to use
Call a program that is to continue an existing logical unit of work.	Set up the parameters as follows: <ul style="list-style-type: none"> • eci_extend_mode: ECI_EXTENDED • eci_program_name: provide it • eci_luw_token: provide it
Call a program that is to be the last program of an existing logical unit of work, and commit the changes.	Set up the parameters as follows: <ul style="list-style-type: none"> • eci_extend_mode: ECI_NO_EXTEND • eci_program_name: provide it • eci_luw_token: provide it
End an existing logical unit of work, without calling another program, and commit changes to recoverable resources.	Set up the parameters as follows: <ul style="list-style-type: none"> • eci_extend_mode: ECI_COMMIT • eci_program_name: null • eci_luw_token: provide it
End an existing logical unit of work, without calling another program, and back out changes to recoverable resources.	Set up the parameters as follows: <ul style="list-style-type: none"> • eci_extend_mode: ECI_BACKOUT • eci_program_name: null • eci_luw_token: provide it

If an error occurs in one of the calls of an extended logical unit of work, you can use the **eci_luw_token** field to see if the changes made so far have been backed out, or are still pending. See the description of the **eci_luw_token** field in “ECI_SYNC call type” on page 19 and “ECI_ASYNC call type” on page 27 for more information. If the changes are still pending, you should end the logical unit of work with another program link call, either committing or backing out the changes.

Each logical unit of work ties up one CICS non-facility task for the duration of its execution. This means that you must define enough free tasks in the CICS server to service the maximum expected number of concurrent calls.

Figure 5 on page 12 shows an application program using asynchronous program calls. The program has two calls outstanding in one server and one call in another. To keep track of its requests, the program is using message qualifiers that it has generated itself, and LUW tokens returned in the **eci_luw_token** field.

External call interface

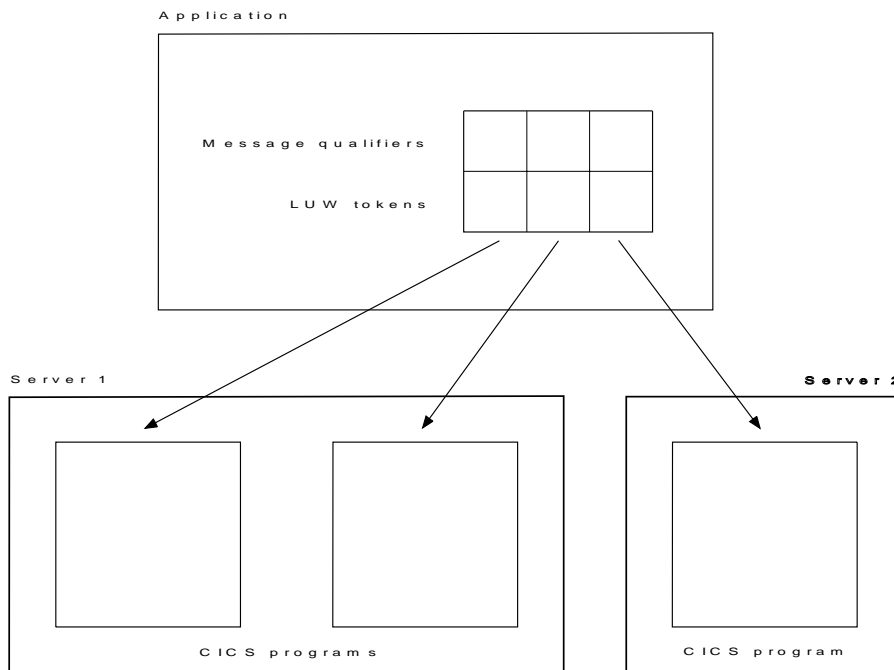


Figure 5. Asynchronous program link calls with message qualifiers and LUW tokens

For more details of the program link calls, you should study the descriptions of the individual call types:

“ECI_SYNC call type” on page 19 for a synchronous program link call

“ECI_ASYNC call type” on page 27 for an asynchronous program link call.

Security in the ECI

The ECI_SYNC and ECI_ASYNC calls are implemented as a set of intersystem conversations between a CICS client and server; a conversation is allocated for each unit of work created by the client application.

A userid and password may be required for each intersystem conversation; the requirement is determined by the way in which the CICS client and CICS server have been configured.

The client application can specify the userid and password in the **eci_userid** (or **eci_userid2**) and **eci_password** (or **eci_password2**) in the ECI parameter block for the first or only call in the unit of work.

The client application can pass the userid and password to the **CICS_SetDefaultSecurity** function. The values are server-specific and are

used whenever the values have been omitted from the ECI parameter block. The values can be changed at any time by the **CICS_SetDefaultSecurity** function.

If the userid and password are required for intersystem conversations and have not been set by the **CICS_SetDefaultSecurity** function then the CICS client may use other methods, for example via a pop-up window, to determine the values. (Note that pop-up windows are not supported on AIX and Solaris.)

DCE protocol considerations

The operator might need DCE authorization to use the ECI application, depending on the local configuration. If the ECI application runs for a long time, for example longer than thirty minutes, you must make sure that the credential lifetime and renewable lifetime of the DCE principal are long enough to cover the maximum allowed duration of the ECI application.

Status information calls

Status information calls can be either synchronous or asynchronous. For asynchronous calls, it is the responsibility of the calling application to obtain the reply using a reply solicitation call (see “Reply solicitation calls” on page 15).

How status information is supplied and used

Status information is supplied in the ECI status block, which is passed across the interface in the **eci_commarea** parameter.

The following status information is held in the ECI status block. For a more detailed description, see “ECI status block” on page 54.

- The type of connection (whether the ECI program is locally connected to a CICS server, a CICS client, or nothing)
- The state of the CICS server (available, unavailable, or unknown)
- The state of the CICS client (available, not applicable, or unknown).

The status information calls allow you to perform three tasks:

- *Enquire on the type of system the application is running on, and its connection with a given server.* For this you need to provide a COMMAREA in which the status is returned.
- *Set up a request to be notified when the status changes from some specified value.* For this you need to provide a COMMAREA in which the specified status

External call interface

is described. When the status is different from the status specified, you are notified of the new status. Only asynchronous calls can be used for this purpose.

- *Cancel a request for notification of status change.* For this no COMMAREA is required.

Table 2 shows how you can use combinations of **eci_extend_mode**, **eci_commarea**, **eci_commarea_length**, and **eci_luw_token** parameter values to perform tasks associated with status enquiries. In each case you must also store appropriate values in other fields for the call type you have chosen.

Table 2. Status enquiries with CICS_ExternalCall

Task to perform	Parameters to use
Find the current status.	Set up the parameters as follows: <ul style="list-style-type: none">• eci_commarea: nulls• eci_commarea_length: length of ECI_STATUS• eci_extend_mode: ECI_STATE_IMMEDIATE Afterwards, consult the contents of the COMMAREA to find the status.
Set up a request to find out about status change.	Set up the parameters as follows: <ul style="list-style-type: none">• eci_commarea: specified status• eci_commarea_length: length of ECI_STATUS• eci_extend_mode: ECI_STATE_CHANGED• eci_luw_token: zero Afterwards, save the token from eci_luw_token so that you can cancel the request later.
Cancel a request to find out about status change.	Set up the parameters as follows: <ul style="list-style-type: none">• eci_commarea: none• eci_commarea_length: zero• eci_extend_mode: ECI_STATE_CANCEL• eci_luw_token: provide it

For more details of the status information calls, you should study the descriptions of the individual call types:

“ECI_STATE_SYNC call type” on page 35 for a synchronous status information call

“ECI_STATE_ASYNC call type” on page 39 for an asynchronous status information call.

Reply solicitation calls

After an asynchronous program link call or asynchronous status information call, it is the responsibility of the calling application to solicit the reply. All calls return any outstanding reply that meets the selection criteria specified in the call.

For more details of the reply solicitation calls, you should study the descriptions of the individual call types:

“ECI_GET_REPLY call type” on page 44 for a reply solicitation call that gets any outstanding reply for any asynchronous call, if any reply is available.

“ECI_GET_REPLY_WAIT call type” on page 48 for a reply solicitation call that gets any outstanding reply for any asynchronous call, waiting if no replies are available.

“ECI_GET_SPECIFIC_REPLY call type” on page 49 for a reply solicitation call that gets any outstanding reply for a given asynchronous call, if any reply is available.

“ECI_GET_SPECIFIC_REPLY_WAIT call type” on page 53 for a reply solicitation call that gets any outstanding reply for a given asynchronous call, waiting if no replies are available.

External call interface

CICS_ExternalCall

CICS_ExternalCall	ECI_Parms
-------------------	-----------

Purpose

CICS_ExternalCall gives access to the program link calls, status information calls, and reply solicitation calls described above. 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. *The parameter block must be set 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 is given next:

eci_call_type

An integer field defining the type of call being made. For details of the functions provided, see “Types of ECI calls” on page 8.

eci_program_name

The name of a program to be called.

eci_userid

User ID for security checking.

eci_password

Password for security checking.

eci_transid

A transaction identifier.

eci_abend_code

Abend code for a failed program.

eci_commarea

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

eci_commarea_length

The length of the COMMAREA. The size of the COMMAREA must be set to the largest size of the input or output data. If the input data is less than the length of the COMMAREA, the COMMAREA should be padded with nulls. The CICS client strips off the null padding and only sends the data on the ECI request to the CICS server.

eci_timeout

The time to wait for a response from the CICS server. For more information on the ECI time-out support, refer to “Time-outs” on page 161.

reserved1

A return code giving more information about an unexpected error.

This field was previously **eci_system_return_code**. In CICS Universal Clients Version 3.1, and higher, this field is reserved for backward compatibility. No information is returned in this field; all system errors are written to the clients error log.

eci_extend_mode

A field that qualifies the function to be performed in various ways.

eci_message_qualifier

A user-provided reference to an asynchronous call.

eci_luw_token

An identifier for a logical unit of work.

eci_sysid

Reserved for future use, leave null.

eci_version

The version of the ECI for which the application is coded. You may use the values `ECI_VERSION_1` or `ECI_VERSION_1A`. All the facilities of version 1 are available in version 1A. Facilities available only in version 1A are noted where they occur. The use of the value `ECI_VERSION_0` is confined to programs migrated from previous versions of the ECI.

eci_system_name

The name of a CICS server.

eci_callback

A pointer to a callback routine for an asynchronous request.

eci_userid2

User ID for security checking.

eci_password2

Password for security checking.

eci_tpn

A transaction identifier for a mirror transaction. This field is available only when **eci_version** has the value `ECI_VERSION_1A`.

External call interface

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_SYSTEM_ERROR

An internal system error occurred. The error might have been in the client or in the server. The programmer should save the information returned in the **eci_sys_return_code** field, as this will help service personnel to diagnose the error.

ECI_ERR_INVALID_VERSION

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

ECI_ERR_REQUEST_TIMEOUT

The value in the **eci_timeout** field of the ECI parameter block is negative.

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 for CICS Universal Client for OS/2 and CICS Universal Client for Windows:

COBOL \copybook\cicseci.cbl

C \include\cics_eci.h

PL/I \plihdr\cics_eci.inc

and in the following files for CICS Universal Client for AIX and CICS Universal Client for Solaris:

COBOL /include/CICSECI

C /include/cics_eci.h

ECI_SYNC call type

Environment

The ECI_SYNC call type is available in all environments.

Purpose

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

ECI parameter block fields

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

eci_call_type

Required input parameter.

Must be set to ECI_SYNC.

eci_program_name

Input parameter, required except when **eci_extend_mode** is ECI_COMMIT or ECI_BACKOUT. (See Table 1 on page 10.)

An 8-character field containing the name of the program to be called. Unused characters should be padded 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, so these should be used with care when communicating with such servers.

eci_userid

Required input parameter.

An 8-character field containing a user ID. Unused characters should be padded with spaces.

You should consult the documentation for the client and the server to check whether this field is converted to upper case before being transmitted to the server. (If a user ID or password longer than 8 characters is required, **eci_version** must not be ECI_VERSION_0; and

External call interface

eci_userid and **eci_password** must be set to nulls, and the fields **eci_userid2** and **eci_password2** used instead.)

If a user ID is supplied, then 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. Unused characters should be padded with spaces.

You should consult the documentation for the client and the server to check whether this field is converted to upper case before being transmitted to the server. (If a user ID or password longer than 8 characters is required, this field and **eci_userid** must be set to nulls, and the fields **eci_userid2** and **eci_password2** used instead.)

eci_transid

Optional input parameter

A 4-character field optionally containing the ID of a CICS transaction. Unused characters should be padded with spaces. This field is ignored if **eci_tpn** is used. This field is transmitted to the server without conversion to uppercase. The use of this parameter depends on the client from which the request is sent.

The called program runs under the mirror transaction CPMT, 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, you are recommended to use this parameter to control the processing of your called programs.

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

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

eci_abend_code

Output parameter.

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

eci_commarea

Optional input parameter.

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

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

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

eci_commarea_length

Optional input parameter.

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

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

eci_timeout

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

If timeout occurs, the conversation is abended.

reserved1

Output parameter.

An integer field containing additional information when the return code is ECI_ERR_SYSTEM_ERROR. The meaning of this information depends on the software and hardware platform. You should save this information, as it will help service personnel to diagnose the error.

This field was previously **eci_system_return_code**. In CICS Universal Clients Version 3.1, and higher, this field is reserved for backward compatibility. No information is returned in this field; all system errors are written to the clients 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 Table 1 on page 10.)

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

ECI_NO_EXTEND

External call interface

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

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

ECI_EXTENDED

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

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

ECI_COMMIT

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

ECI_BACKOUT

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

eci_luw_token

Required input and output parameter.

An integer field used for identifying the logical unit of work to which a call belongs. It must be set to zero at the start of a logical unit of work (regardless of whether the logical unit of work is going to be extended). If the logical unit of work is to be extended, the ECI updates **eci_luw_token** with a valid value on the first call of the logical unit of work, and this value should be used as input to all later calls related to the same logical unit of work. (See Table 1 on page 10.)

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

eci_sysid

Required input parameter.

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

eci_version

Required input parameter.

The version of the ECI for which the application is coded. You may use the values `ECI_VERSION_1` or `ECI_VERSION_1A`. All the facilities of version 1 are available in version 1A. Facilities available only in version 1A are noted where they occur. (`ECI_VERSION_0` is provided to allow applications written for previous versions of the ECI to continue to execute, but many of the facilities of CICS Client/Server Programming are not available if **eci_version** has this value.)

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. Unused characters should be padded with spaces. If supplied, it should be one of the server names returned by **CICS_EciListSystems**. The value may be supplied whenever **eci_luw_token** is set to zero. (If it is supplied when **eci_luw_token** is not zero, it is ignored, because the server was established at the start of the logical unit of work.)

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

eci_userid2

Optional input parameter.

If the **eci_userid** field is set to nulls, then 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. **eci_version** must not be `ECI_VERSION_0`.

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. **eci_version** must not be `ECI_VERSION_0`.

External call interface

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

eci_tpn

Optional input parameter. Available only if the value of **eci_version** is **ECI_VERSION_1A**.

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

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

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

Return codes

See also the general list of return codes for **CICS_ExternalCall** in “CICS_ExternalCall” on page 16.

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

ECI_ERR_CICS_DIED

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

ECI_ERR_TRANSACTION_ABEND

The CICS transaction that executed the requested program abended. The abend code will be found in **eci_abend_code**. For information about abend codes and their meaning, you should 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 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_INVALID_TRANSID

A logical unit of work was being extended, but the value supplied in **eci_transid** differed from the value used when the logical unit of work was started.

ECI_ERR_MAX_SESSIONS

There were not enough communication resources to satisfy the request. You should consult the documentation for your client or server to see how to control communication resources.

ECI_ERR_MAX_SYSTEMS

You attempted to start requests to more servers than your configuration allows. You should consult the documentation for your client or server to see how to control the number of servers you can use.

External call interface

ECI_ERR_SECURITY_ERROR

You did not supply a valid combination of user ID and password, though the server expects it.

ECI_ASYNC call type

Environment

The ECI_ASYNC call type is available in all environments except DOS.

Purpose

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

If no callback routine is provided, the application must use a reply solicitation call to determine 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.

When the callback routine is called, it is passed a single parameter—the value specified in **eci_message_qualifier**. Use of this parameter enables the callback routine to identify the asynchronous call that is completing. Note the following guidelines on the use of the callback routine:

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

ECI parameter block fields

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

External call interface

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 Table 1 on page 10.)

An 8-character field containing the name of the program to be called. Unused characters should be padded 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, so these should be used with care when communicating with such servers.

eci_userid

Required input parameter.

An 8-character field containing a user ID. Unused characters should be padded with spaces.

You should consult the documentation for the client and the server to check whether this field is converted to upper case before being transmitted to the server. (If a user ID or password longer than 8 characters is required, **eci_version** must not be ECI_VERSION_0; and **eci_userid** and **eci_password** must be set to nulls, and the fields **eci_userid2** and **eci_password2** used instead.)

If a user ID is supplied, then the user ID and any supplied password will be used by the server to validate the authority of the user to execute the ECI request. The form of this validation is defined by the server.

eci_password

Required input parameter.

An 8-character field containing a password. Unused characters should be padded with spaces.

You should consult the documentation for the client and the server to check whether this field is converted to upper case before being transmitted to the server. (If a user ID or password longer than 8 characters is required, this field and **eci_userid** must be set to nulls, and the fields **eci_userid2** and **eci_password2** used instead.)

eci_transid

Optional input parameter.

A 4-character field optionally containing a CICS transaction ID. Unused characters should be padded with spaces. This field is ignored if **eci_tpn** is used. This field is transmitted to the server without conversion to uppercase. The use of this parameter depends on the client from which the request is sent.

The called program runs under the mirror transaction CPMT, 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, you are recommended to use this parameter to control the processing of your called programs.

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

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

eci_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, you need to make use of CICS-supplied resource conversion capabilities, such as the DFHCNV macro definitions.

eci_commarea_length

Required input parameter.

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

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.

External call interface

If timeout occurs, the conversation is abended.

reserved1

Output parameter.

An integer field containing additional information when the return code is ECI_ERR_SYSTEM_ERROR. The meaning of this information depends on the software and hardware platform. You should save this information, as it will help service personnel to diagnose the error.

This field was previously **eci_system_return_code**. In CICS Universal Clients Version 3.1, and higher, this field is reserved for backward compatibility. No information is returned in this field; all system errors are written to the clients 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 Table 1 on page 10.)

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

ECI_NO_EXTEND

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

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

ECI_EXTENDED

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

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

ECI_COMMIT

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

ECI_BACKOUT

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

eci_message_qualifier

Optional input parameter.

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

eci_luw_token

Required input and output parameter.

An integer field used for identifying the logical unit of work to which a call belongs. It must be set to zero at the start of a logical unit of work (regardless of whether the logical unit of work is going to be extended), and the ECI updates it with a valid value on the first or only call of the logical unit of work. If the logical unit of work is to be extended, this value should be used as input to all later calls related to the same logical unit of work. (See Table 1 on page 10.)

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

eci_sysid

Required input parameter.

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

eci_version

Required input parameter.

The version of the ECI for which the application is coded. You may use the values ECI_VERSION_1 or ECI_VERSION_1A. All the facilities of version 1 are available in version 1A. Facilities available only in version 1A are noted where they occur. (ECI_VERSION_0 is provided to allow applications written for previous versions of the ECI to continue to execute, but many of the facilities of CICS Client/Server Programming are not available if **eci_version** has this value.)

eci_system_name

Optional input parameter.

External call interface

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

If the field is set to nulls, then a server, currently the default server, is selected. You can obtain the name of the chosen server from the **eci_system_name** field of the reply solicitation call you use to get the result of this asynchronous request. (If later ECI requests made in different logical units of work must be directed to the same server as this request, then **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, then 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. **eci_version** must not be **ECI_VERSION_0**.

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. **eci_version** must not be **ECI_VERSION_0**.

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

eci_tpn

Optional input parameter. Available only if the value of **eci_version** is **ECI_VERSION_1A**.

A 4-character field that specifies the transaction ID of the transaction that will be used in the server to process the ECI request. This

transaction must be defined in the server as a CICS mirror transaction. If the field is not set, the default mirror transaction CPMI is used.

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

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

Return codes

See also the general list of return codes for **CICS_ExternalCall** in “CICS_ExternalCall” on page 16.

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.

External call interface

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_INVALID_TRANSID

A logical unit of work was being extended, but the value supplied in **eci_transid** differed from the value used when the logical unit of work was started.

ECI_STATE_SYNC call type**Environment**

The ECI_STATE_SYNC call type is available in all environments.

Purpose

The ECI_STATE_SYNC call type provides a synchronous status information call.

ECI parameter block fields

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

eci_call_type

Required input parameter.

Must be set to ECI_STATE_SYNC.

eci_commarea

Input parameter, required except when **eci_extend_mode** has the value ECI_STATE_CANCEL.

A pointer to the area of storage where the application receives the returned COMMAREA containing status information (see “Status information calls” on page 13 and “ECI status block” on page 54 for more details).

If **eci_extend_mode** has the value ECI_STATE_CANCEL, supply a null pointer and set the length (specified in **eci_commarea_length**) to zero.

eci_commarea_length

Required input and output parameter, except when **eci_extend_mode** has the value ECI_STATE_CANCEL.

The length of the COMMAREA in bytes, which must be the length of the ECI_STATUS structure that gives the layout of the status information COMMAREA (see “Status information calls” on page 13 and “ECI status block” on page 54 for more details).

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

reserved1

Output parameter.

An integer field containing additional information when the return code is ECI_ERR_SYSTEM_ERROR. The meaning of this information

External call interface

depends on the software and hardware platform. You should save this information, as it will help service personnel to diagnose the error.

This field was previously **eci_system_return_code**. In CICS Universal Clients Version 3.1, and higher, this field is reserved for backward compatibility. No information is returned in this field; all system errors are written to the clients error log.

eci_extend_mode

Required input parameter.

An integer field further qualifying the call type. (See Table 2 on page 14.) The values for this field (shown by their symbolic names) are as follows:

ECI_STATE_IMMEDIATE

Force a status reply to be sent immediately it is available. The layout of the returned COMMAREA is defined in the ECI_STATUS structure (see “Status information calls” on page 13 and “ECI status block” on page 54).

ECI_STATE_CHANGED

Force a status reply to be sent only when the status changes. The supplied COMMAREA must contain the status as perceived by the application. A reply is sent only when there is a change from the status that the application supplied. The layout of the COMMAREA is defined in the ECI_STATUS structure (see “Status information calls” on page 13 and “ECI status block” on page 54 for more details). The **eci_luw_token** field that is returned on the immediate response provides a token to identify the request.

ECI_STATE_CANCEL

Cancel an ECI_STATE_CHANGED type of operation. No COMMAREA is required for this request. The **eci_luw_token** field must contain the token that was received during the ECI_STATE_CHANGED call.

eci_luw_token

Optional input and output parameter.

When a deferred status request is being set up (**eci_extend_mode** set to ECI_STATE_CHANGED), the token identifying the request is returned in the **eci_luw_token** field.

When a deferred status request is being cancelled (**eci_extend_mode** set to ECI_STATE_CANCEL), the **eci_luw_token** field must contain the token that was received during the ECI_STATE_CHANGED call.

This field is not used when other values of **eci_extend_mode** are specified.

eci_sysid

Required input parameter.

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

eci_version

Required input parameter.

The version of the ECI for which the application is coded. You may use the values `ECI_VERSION_1` or `ECI_VERSION_1A`. All the facilities of version 1 are available in version 1A. Facilities available only in version 1A are noted where they occur. (`ECI_VERSION_0` is provided to allow applications written for previous versions of the ECI to continue to execute, but many of the facilities of CICS Client/Server Programming are not available if `eci_version` has this value.)

eci_system_name

Optional input parameter.

An 8-character field that specifies the name of the server for which status information is required. Unused characters should be padded with spaces. If supplied, it should be one of the server names returned by `CICS_EciListSystems`. The value may be supplied whenever `eci_luw_token` is set to zero.

If the field is set to nulls, then a server, currently the default server, is selected; the name of the chosen server is returned in this

Return codes

See also the general list of return codes for `CICS_ExternalCall` in “`CICS_ExternalCall`” on page 16.

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.

External call interface

ECI_ERR_INVALID_DATA_AREA

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

ECI_ERR_UNKNOWN_SERVER

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

ECI_STATE_ASYNC call type

Environment

The ECI_STATE_ASYNC call type is available in all environments except DOS.

Purpose

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.

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.

When the callback routine is called, it is passed a single parameter—the value specified in **eci_message_qualifier**. Use of this parameter enables the callback routine to identify the asynchronous call that is completing. Note the following guidelines on the use of the callback routine:

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

ECI parameter block fields

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

External call interface

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 “Status information calls” on page 13 and “ECI status block” on page 54 for more details).

If **eci_extend_mode** has the value `ECI_STATE_CANCEL`, supply a null pointer and set the length (specified in **eci_commarea_length**) to zero.

eci_commarea_length

Required input parameter, except when **eci_extend_mode** has the value `ECI_STATE_CANCEL`.

The length of the COMMAREA in bytes, which must be the length of the `ECI_STATUS` structure that gives the layout of the status information COMMAREA (see “Status information calls” on page 13 and “ECI status block” on page 54 for more details).

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

reserved1

Output parameter.

An integer field containing additional information when the return code is `ECI_ERR_SYSTEM_ERROR`. The meaning of this information depends on the software and hardware platform. You should save this information, as it will help service personnel to diagnose the error.

This field was previously **eci_system_return_code**. In CICS Universal Clients Version 3.1, and higher, this field is reserved for backward compatibility. No information is returned in this field; all system errors are written to the clients error log.

eci_extend_mode

Required input parameter.

An integer field further qualifying the call type. (See Table 2 on page 14.) The values for this field (shown by their symbolic names) are as follows:

ECI_STATE_IMMEDIATE

Force a status reply to be sent immediately it is available. The

layout of the returned COMMAREA is defined in the ECI_STATUS structure (see “Status information calls” on page 13 and “ECI status block” on page 54

ECI_STATE_CHANGED

Force a status reply to be sent only when the status changes. The supplied COMMAREA must contain the status as perceived by the application. A reply is sent only when there is a change from the status that the application supplied. The layout of the COMMAREA is defined in the ECI_STATUS structure (see “Status information calls” on page 13 and “ECI status block” on page 54 for more details). The **eci_luw_token** field that is returned on the immediate response identifies the logical unit of work to which this call belongs.

ECI_STATE_CANCEL

Cancel an ECI_STATE_CHANGED type of operation. No COMMAREA is required for this request. The **eci_luw_token** field must contain the token that was received during the ECI_STATE_CHANGED call.

eci_message_qualifier

Optional input parameter.

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

eci_luw_token

Optional input and output parameter.

When a deferred status request is being set up (**eci_extend_mode** set to ECI_STATE_CHANGED), the token identifying the request is returned in the **eci_luw_token** field.

When a deferred status request is being cancelled (**eci_extend_mode** set to ECI_STATE_CANCEL), the **eci_luw_token** field must contain the token that was received during the ECI_STATE_CHANGED call.

This field is not used when other values of **eci_extend_mode** are specified.

eci_sysid

Required input parameter.

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

eci_version

Required input parameter.

External call interface

The version of the ECI for which the application is coded. You may use the values `ECI_VERSION_1` or `ECI_VERSION_1A`. All the facilities of version 1 are available in version 1A. Facilities available only in version 1A are noted where they occur. (`ECI_VERSION_0` is provided to allow applications written for previous versions of the ECI to continue to execute, but many of the facilities of CICS Client/Server Programming are not available if `eci_version` has this value.)

eci_system_name

Optional input parameter.

An 8-character field that specifies the name of the server for which status information is requested. Unused characters should be padded with spaces. If supplied, it should be one of the server names returned by `CICS_EciListSystems`. The value may be supplied whenever `eci_luw_token` is set to zero.

If the field is set to nulls, then a server, currently the default server, is selected. You can find out the name of the server from the `eci_system_name` field of the reply solicitation call you use to get the result of this asynchronous request. field.

eci_callback

Optional input parameter.

A pointer to the routine to be called when the asynchronous request completes. (The callback routine will be called only if the return code is `ECI_NO_ERROR`, and the pointer is not null.)

Return codes

See also the general list of return codes for `CICS_ExternalCall` in “CICS_ExternalCall” on page 16.

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.

External call interface

ECI_GET_REPLY call type

Environment

The ECI_GET_REPLY call type is available in all environments except DOS.

Purpose

The ECI_GET_REPLY call type provides a reply solicitation call to return information appropriate to any outstanding reply for any asynchronous request. If there is no such reply, ECI_ERR_NO_REPLY is returned. (To cause the application to wait until a reply is available, you should use call type ECI_GET_REPLY_WAIT instead.)

ECI parameter block fields

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

The following fields are the fields of the ECI parameter block that might be supplied as input.

In the course of an ECI_GET_REPLY call, the ECI parameter block is updated as follows:

1. All the outputs from the reply, some of which overwrite input fields, are added. These fields are those that are output from the corresponding synchronous version of the asynchronous request.
2. The **eci_message_qualifier** value supplied as input to the asynchronous request to which this reply relates is restored.
3. Any inputs that are not updated become undefined, except the pointer to the COMMAREA. You should not use the contents of these fields again.

eci_call_type

Required input parameter.

Must be set to ECI_GET_REPLY.

eci_commarea

Optional input parameter.

A pointer to the area of storage where the application receives the returned COMMAREA. The contents of the returned commarea depend on the type of asynchronous call to which a reply is being sought. For a program link call, it is the COMMAREA expected to be returned from the called program, if any. For a status information call, except when **eci_extend_mode** has the value ECI_STATE_CANCEL, it

is a COMMAREA containing status information (see “Status information calls” on page 13 and “ECI status block” on page 54 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, you need to make use of CICS-supplied resource conversion capabilities, such as the DFHCNV macro definitions.

eci_commarea_length

Required input parameter.

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

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

eci_sysid

Required input parameter.

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

eci_version

Required input parameter.

The version of the ECI for which the application is coded. You may use the values **ECI_VERSION_1** or **ECI_VERSION_1A**. All the facilities of version 1 are available in version 1A. Facilities available only in version 1A are noted where they occur. (**ECI_VERSION_0** is provided to allow applications written for previous versions of the ECI to continue to execute, but many of the facilities of CICS Client/Server Programming are not available if **eci_version** has this value.)

Return codes

See also the general list of return codes for **CICS_ExternalCall** in “CICS_ExternalCall” on page 16.

ECI_NO_ERROR

The asynchronous request to which this reply relates completed successfully.

External call interface

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, you should consult the documentation for the server system to which the request was directed.

ECI_ERR_THREAD_CREATE_ERROR

The CICS server or client failed to create the thread to process the asynchronous call to which this reply relates.

ECI_ERR_RESOURCE_SHORTAGE

The server implementation or client did not have enough resources to complete the asynchronous request to which this reply relates.

ECI_ERR_INVALID_DATA_AREA

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

ECI_ERR_ROLLEDBACK

The asynchronous request to which this reply relates attempted to commit a logical unit of work, but the server was unable to commit the changes, and backed them out instead.

ECI_ERR_UNKNOWN_SERVER

The asynchronous request to which this reply relates specified a server that could not be located. Only servers returned by `CICS_EciListSystems` are acceptable.

ECI_ERR_MAX_SESSIONS

There were not enough communication resources to satisfy the asynchronous request to which this reply relates. You should consult the documentation for your client or server to see how to control communication resources.

ECI_ERR_MAX_SYSTEMS

The asynchronous request to which this reply relates attempted to start requests to more servers than your configuration allows. You should consult the documentation for your client or server to see how to control the number of servers you can use.

ECI_ERR_SECURITY_ERROR

You did not supply a valid combination of `userid` and `password` on the asynchronous request to which this reply relates, though the server expects it.

External call interface

ECI_GET_REPLY_WAIT call type

Environment

The ECI_GET_REPLY_WAIT call type is available in all environments except DOS.

Purpose

The ECI_GET_REPLY_WAIT call type provides a reply solicitation call to return information appropriate to any outstanding reply for any asynchronous request. If there is no such reply, the application waits until there is. (In all environments except DOS, you can get an indication that no reply is available by using call type ECI_GET_REPLY instead.)

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

Environment

The ECI_GET_SPECIFIC_REPLY call type is available in all environments except DOS.

Purpose

The ECI_GET_SPECIFIC_REPLY call type provides a reply solicitation call to return information appropriate to any outstanding reply that matches the **eci_message_qualifier** input. If there is no such reply, ECI_ERR_NO_REPLY is returned. (To cause the application to wait until a reply is available, you should use call type ECI_GET_REPLY_WAIT instead.)

ECI parameter block fields

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

The following fields are the fields of the ECI parameter block that might be supplied as input.

In the course of an ECI_GET_REPLY call, the ECI parameter block is updated as follows:

1. All the outputs from the reply, some of which overwrite input fields, are added. These fields are those that are output from the corresponding synchronous version of the asynchronous request.
2. Any inputs that are not updated become undefined, except the pointer to the COMMAREA and the input **eci_message_qualifier**. You should 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

External call interface

information (see “Status information calls” on page 13 and “ECI status block” on page 54 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, you need to make use of CICS-supplied resource conversion capabilities, such as the DFHCNV macro definitions.

eci_commarea_length

Required input parameter.

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

eci_message_qualifier

Required input parameter.

An integer field that identifies the asynchronous call for which a reply is being solicited.

eci_sysid

Required input parameter.

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

eci_version

Required input parameter.

The version of the ECI for which the application is coded. You may use the values ECI_VERSION_1 or ECI_VERSION_1A. All the facilities of version 1 are available in version 1A. Facilities available only in version 1A are noted where they occur. (ECI_VERSION_0 is provided to allow applications written for previous versions of the ECI to continue to execute, but many of the facilities of CICS Client/Server Programming are not available if **eci_version** has this value.)

Return codes

See also the general list of return codes for **CICS_ExternalCall** in “CICS_ExternalCall” on page 16.

ECI_NO_ERROR

The call completed successfully.

ECI_ERR_INVALID_DATA_LENGTH

The value in **eci_commarea_length** field is unacceptable for one of the following reasons:

- It is outside the valid range.
- It is inconsistent with the value in `eci_commarea`, being zero for a non-null `eci_commarea` pointer, or non-zero for a null `eci_commarea` pointer.
- It is not large enough for the output COMMAREA from the asynchronous request to which this reply relates.

In the last case, you can use the output `eci_commarea_length` to allocate more storage for the COMMAREA, and then retry the `ECI_GET_SPECIFIC_REPLY` call.

ECI_ERR_NO_CICS

The CICS server specified in `eci_system_name` in the asynchronous request to which this reply relates is not available. No resources have been updated.

ECI_ERR_CICS_DIED

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

ECI_ERR_NO_REPLY

There was no outstanding reply that matched the input `eci_message_qualifier`.

ECI_ERR_TRANSACTION_ABEND

The asynchronous request to which this reply relates caused a program to be executed in the server, but the CICS transaction that executed the requested program abended. The abend code will be found in `eci_abend_code`. For information about abend codes and their meaning, you should consult the documentation for the server system to which the request was directed.

ECI_ERR_THREAD_CREATE_ERROR

The CICS server or client failed to create the thread to process the asynchronous request to which this reply relates.

ECI_ERR_RESOURCE_SHORTAGE

The CICS server or client 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.

External call interface

ECI_ERR_ROLLEDBACK

The asynchronous request to which this reply relates attempted to commit a logical unit of work, but the server was unable to commit the changes, and backed them out instead.

ECI_ERR_UNKNOWN_SERVER

The asynchronous request to which this reply relates specified a server that could not be located. Only servers returned by **CICS_EciListSystems** are acceptable.

ECI_ERR_MAX_SESSIONS

There were not enough communication resources to satisfy the asynchronous request to which this reply relates. You should consult the documentation for your client or server to see how to control communication resources.

ECI_ERR_MAX_SYSTEMS

The asynchronous request to which this reply relates attempted to start requests to more servers than your configuration allows. You should consult the documentation for your client or server to see how to control the number of servers you can use.

ECI_ERR_SECURITY_ERROR

You did not supply a valid combination of userid and password on the asynchronous request to which this reply relates, though the server expects it.

ECI_GET_SPECIFIC_REPLY_WAIT call type

Environment

The ECI_GET_SPECIFIC_REPLY_WAIT call type is available in all environments except DOS.

Purpose

The ECI_GET_SPECIFIC_REPLY_WAIT call type provides a reply solicitation call to return information appropriate to any outstanding reply that matches the input **eci_message_qualifier**. If there is no such reply, the application waits until there is. (In all environments except DOS, you can get an indication that no reply is available by using call type ECI_GET_SPECIFIC_REPLY instead.)

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.

External call interface

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, with the following possible values:

`ECI_CLIENTSTATE_UNKNOWN`

The client state could not be determined.

`ECI_CLIENTSTATE_UP`

The client is available to receive ECI calls.

`ECI_CLIENTSTATE_INAPPLICABLE`

The application is using a server implementation of the ECI.

CICS_EciListSystems

CICS_EciListSystems	NameSpace Systems List
---------------------	---

Purpose

The **CICS_EciListSystems** function provides a list of CICS servers to which **CICS_ExternalCall** requests may be directed. There is no guarantee that a communications link exists between the client and any server in the list, or that any of the servers is available to process requests.

The list of servers is returned as an array of system information structures, one element for each CICS server. The structure is called **CICS_EciSystem_t**, and it defines the following fields.

SystemName

A pointer to a null-terminated string specifying the name of a CICS server. If the name is shorter than **CICS_ECI_SYSTEM_MAX**, it is padded with nulls to a length of **CICS_ECI_SYSTEM_MAX + 1**.

Description

A pointer to a null-terminated string that provides a description of the system, if one is available. If the description is shorter than **CICS_ECI_DESCRIPTION_MAX** characters, it is padded with nulls to a length of **CICS_ECI_DESCRIPTION_MAX + 1**.

Parameters

NameSpace

A pointer reserved for future use. Ensure that this is a null pointer.

Systems

On entry to the function, this parameter specifies the number of elements in the array provided in the **List** parameter. On return it contains the actual number of systems found.

List

An array of **CICS_EciSystem_t** structures that are filled in and returned by the function. The application must provide the storage for the array, and must set the **Systems** parameter to indicate the number of elements in the array. The first name in the list is the default server. However, the way in which the default is defined is operating system dependent.

ECI status block

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 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.

Chapter 3. External presentation interface

This chapter provides reference information about the external presentation interface (EPI).

The interface is described here in a language-independent manner, though the names chosen for the elements of the interface—functions, parameters, data structures, fields, constants, and so on—are similar to those provided for programming. For language-dependent information, see “Chapter 4. Creating ECI and EPI application programs” on page 127.

Any restrictions applicable to particular operating environments are identified under the functions to which they apply.

The chapter is organized as follows:

“Overview”

“How to use the EPI” on page 58

“EPI constants and data structures” on page 67

“EPI functions” on page 77

“EPI events” on page 114

“3270 data streams for the EPI” on page 121

“Microsoft Windows 3.1 considerations” on page 125.

Overview

The EPI allows a non-CICS application program to have access to one or more CICS servers. In each CICS server, the application can establish one or more terminal resources. The application acts as the operator of these terminal resources, starting CICS transactions, and sending and receiving standard 3270 data streams associated with those transactions.

The following points should be borne in mind when designing client/server implementations using the EPI.

1. A CICS transaction that sends data to an EPI application must not use basic mapping support (BMS) paging.
2. The EPI application can use the EPI to determine the default screen size of the terminal resource definition, but not the alternate screen size.

External presentation interface

3. The EPI application cannot use the EPI to determine whether it is to send or receive double byte character set (DBCS) data streams to or from the CICS transaction.
4. The 3270 data streams produced by CICS transactions must not contain structured fields.
5. For EPI terminals on CICS Universal Clients, the maximum screen size is 27 rows by 132 columns. This is because CICS Universal Clients support the ASCII-7 subset of the 3270 data stream architecture, which only supports 12 bit addressing. For more information, see “3270 data streams for the EPI” on page 121.
6. If a CICS transaction is run with execution diagnostic facility (EDF) enabled at an EPI application’s terminal resource, the events reported to the EPI application are different from those reported when EDF is not enabled.
7. If a CICS transaction uses EXEC CICS START with the DELAY option to schedule transactions to a terminal resource autoinstalled by an EPI application, the EPI application should exercise caution in deleting the terminal resource, or delayed ATI requests might be lost. You should consult your server documentation to determine the effects of deleting a terminal resource when delayed ATI requests are outstanding.

The application is responsible for the presentation of the 3270 data received. The application may present the data in the normal way by emulating a 3270 terminal, or it may present a very different view. For example:

- An OS/2 application may use the OS/2 graphical user interface.
- A Windows NT application may use the Windows NT graphical user interface.
- A SunOS or Solaris application may use Open Look.

How to use the EPI

The EPI provides a separate function name for each of its functions.

Initialization and termination

Any application requiring to use the EPI must call the **CICS_EpiInitialize** function to initialize the EPI. Until this call is made, no other EPI function is allowed. The **CICS_EpiInitialize** function takes a parameter indicating the version of the EPI for which the application was coded. This is to ensure that existing applications continue to execute without change if the EPI is extended.

Before an EPI application ends, it should call the **CICS_EpiTerminate** function to terminate the EPI cleanly.

Listing the configured servers

After calling **CICS_EpiInitialize** successfully, the application should call **CICS_EpiListSystems** to check which CICS servers are configured for use by the application.

Adding terminal resources

The primary purpose of the EPI is to allow an application to appear to a CICS server as one or more 3270 terminals.

Terminal resources are added by invoking either the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** function.

The application may specify the server in which the terminal resource is to be installed; if it does not, a server, currently the default server, is selected.

The application may specify the terminal resource to be installed by supplying one of the following:

- The netname of an existing terminal resource in the server. This terminal resource should be one that is not currently in use.
- The name of a model terminal definition. In this case, the server generates a netname and autoinstalls a terminal resource with that netname, using the specified model terminal definition.
- Nothing. In this case the server generates a netname and autoinstalls a terminal resource with that name, using a default model terminal definition.

Once an application has installed a terminal resource, no other application can use that terminal resource until the first application deletes it.

The **CICS_EpiAddTerminal** and **CICS_EpiAddExTerminal** functions return a *terminal index*, which must be passed on subsequent EPI function calls to indicate the terminal resource to which the function is to apply. Each index identifies a combination of server and terminal resource. The terminal index supplied is the first available integer starting from 0. This allows the application to maintain a mapping between terminal indexes and terminal resource information, using array lookup.

Terminal indexes are unique within an application, but not across applications, so each application gets terminal index zero for the first terminal it installs, and so on.

Figure 6 on page 60 shows an application with three terminal resources, one in one server and two in another. The application uses the terminal indexes to operate the terminal resources, sending and receiving 3270 data streams. The

External presentation interface

order of the indexes shows the order in which the terminal resources were installed.

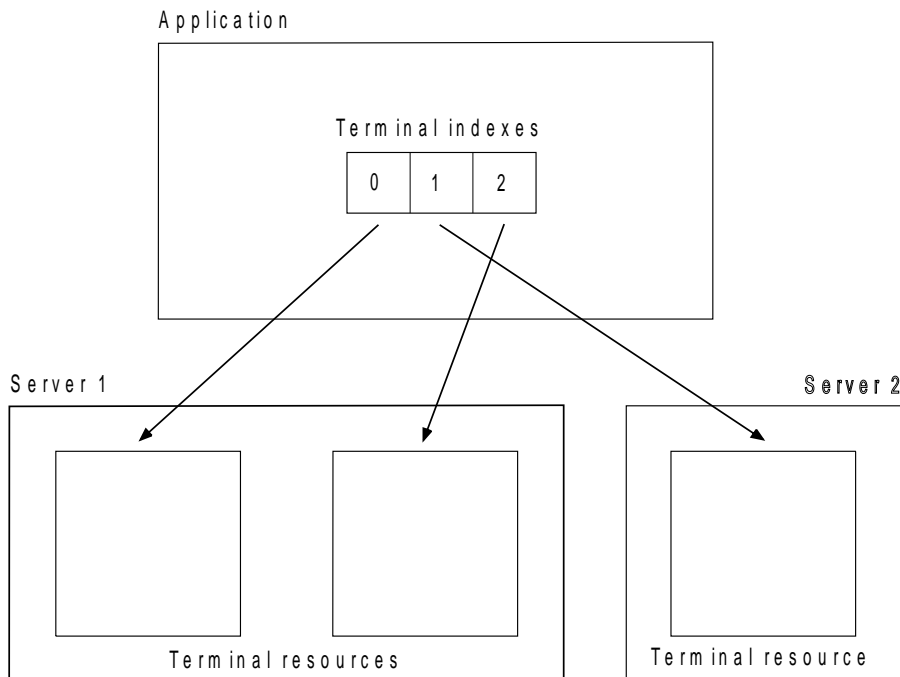


Figure 6. Using terminal indexes

Terminal attributes

Most attributes are determined by the server when the terminal resource is installed; a subset are returned to the application in the **CICS_EpiDetails_t** structure passed to the **CICS_EpiAddTerminal** and **CICS_AddExTerminal** functions.

Some attributes may be determined by the application. These attributes are specified in the **CICS_EpiAttributes_t** structure passed to the **CICS_EpiAddExTerminal** function. For example the character set and encoding scheme to be used for 3270 data is specified in the **CCSID** field in the **CICS_EpiAttributes_t** structure.

Timeout

Execution of the **CICS_EpiAddTerminal** function is synchronous; execution of the **CICS_EpiAddExTerminal** function may be synchronous. Control is not returned to the invoking application until a response, for example, that the terminal resource has been installed, is available

The response time depends on a number of factors; for example:

- Network parameters and traffic
- Server parameters and transaction load

The application is able to limit the length of time that it is prepared to wait for a response; the limit is specified in the **InstallTimeOut** field in the **CICS_EpiAttributes_t** structure passed to the **CICS_EpiAddExTerminal** function.

If the client does not receive a response from the server within the specified interval, control will be returned to the invoking application with the return code set to **CICS_EPI_ERR_RESPONSE_TIMEOUT**.

If the client is subsequently notified that the terminal resource has been installed in the server, the client will delete the terminal resource. Note that this action has the effect of increasing the network traffic and the transaction load on the server.

Deleting terminal resources

If a terminal resource is no longer required, it can be deleted by invoking either the **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal** function.

The **CICS_EpiDelTerminal** function is used when the terminal resource is to be deleted in a controlled manner. The call succeeds only if no transaction is running against the terminal resource and there are no unprocessed events for the terminal resource; if a transaction is in progress or there are unprocessed events, the call returns an error code.

The **CICS_EpiPurgeTerminal** function is used when the terminal resource is to be deleted without regard to a transaction that may be running against the terminal or unprocessed events for the terminal resource.

When the terminal resource has been deleted the terminal index value becomes free and can be reused when another terminal resource is added. The server deletes the terminal resource if it was autoinstalled.

Authentication and authorization

When an application has defined a terminal resource, it can start transactions from that terminal resource. The server may have to:

- Authenticate the userid and password for the terminal "operator"
- Grant authority, based on the authenticated userid, to access the resources required for the execution of each transaction.

External presentation interface

The frequency with which the userid and password are authenticated by the server depends on another attribute specified in the **SignOnCapability** field in the **CICS_EpiAttributes_t** structure passed to the **CICS_EpiAddExTerminal** function.

Signon capable terminals

If the terminal resource is installed as signon capable, the application is responsible for starting a signon transaction, for example the CICS supplied CESN transaction; the userid and password being embedded in the 3270 data. Transactions started before the signon transaction will execute with the authorities granted to a default userid. Transactions started after the signon transaction will execute with the authorities granted to the authenticated userid.

A consequence of installing a terminal as signon capable is that the application must be prepared to cope with the "operator" being signed off if the terminal resource remains idle for longer than a server defined limit.

Signon incapable terminals

If the terminal resource is installed as signon incapable, the userid and password are authenticated for each transaction started for the terminal resource. The initial userid and password for the terminal resource may be specified in the **UserId** and **Password** fields in the **CICS_EpiAttributes_t** structure passed to the **CICS_EpiAddExTerminal** function.

The userid and password may be changed at any time by invoking the **CICS_EpiSetSecurity** function.

If the userid and password are required and are not provided by the application, the CICS client may use, for example, a pop-up window to obtain the userid and password (Note that pop-up windows are not supported on AIX and Solaris).

Starting transactions

When an application has defined a terminal resource, it can start a transaction from that terminal resource. To the CICS server it appears as if a terminal user had typed a transaction code on the screen and pressed an AID key. To start a transaction, the **CICS_EpiStartTran** function is called. There are two ways of specifying the transaction to be started and the data to be associated with it.

1. Supply the transaction identifier as a parameter to the call (**TransId**), and supply any transaction data in the **Data** parameter.
2. Combine a transaction identifier and transaction data into a 3270 data stream, and supply the data stream as a parameter to the call (**Data**).

Events and callbacks

When an application has defined a terminal resource, several events can happen to it because of actions in the CICS server, rather than actions of the application. The application cannot predict when those events may happen, but it is the responsibility of the application to collect and process the events as appropriate. Events are held by the EPI in a first-in-first-out queue until they are collected, one by one, by the application making calls to the **CICS_EpiGetEvent** function. During such a call, the EPI puts information in a **CICS_EpiEventData_t** structure to indicate the event that occurred and any associated data. It also indicates whether there are more events still waiting in the queue.

The application can synchronize the processing of these events with its other activities in one of three ways:

- Polling
- Blocking
- Callback notification.

Polling

The **CICS_EpiGetEvent** call can be made in a polling mode by specifying **CICS_EPI_NOWAIT** for the **Wait** parameter. If no event is waiting to be collected, the function returns immediately with an error code. This is the mechanism that you would have to adopt in a single-user single-threaded environment, such as DOS, where the application might alternately poll the keyboard for user activity and poll the EPI for event activity.

Blocking

The **CICS_EpiGetEvent** call can be made in a blocking mode by specifying **CICS_EPI_WAIT** for the **Wait** parameter. If no event is waiting to be collected, the function waits and does not return until an event becomes available. You could use this mechanism in a multithreaded environment, where a secondary thread could be dedicated to event processing. It could also be used after a notification by callback, because the event information is known to be available.

Callback notification

When the terminal resource is defined with the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call, the optional parameter **NotifyFn** may be used to provide the address of a callback routine that the EPI calls whenever an event occurs against that terminal resource. Callback notification is not

External presentation interface

supported under DOS and Microsoft Windows. Under DOS and Microsoft Windows the parameter can be specified, but it is ignored, and the application must poll for events.

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

An application should carry out the minimum of processing in its callback routine, and never block in the specified routine before returning to the EPI. The routine itself cannot make EPI calls. You decide what it should do when the notification is received. For example, in a multithreaded environment like OS/2, it might post a semaphore to signal another thread that an event has occurred. In a Presentation Manager environment, it might post a message to a window to indicate to the window procedure that an event has occurred. Other actions will be appropriate for other environments.

When the callback routine is called, it is passed a single parameter—the terminal index of the terminal resource against which the event occurred. This allows the same callback routine to be used for more than one terminal resource.

Processing the events

On return from the **CICS_EpiGetEvent** function, the **CICS_EpiEventData_t** structure contains the details of the event that occurred. The **Event** field in this structure indicates the event, and can take one of the following values:

- CICS_EPI_EVENT_SEND
- CICS_EPI_EVENT_CONVERSE
- CICS_EPI_EVENT_END_TRAN
- CICS_EPI_EVENT_START_ATI
- CICS_EPI_EVENT_ADD_TERM
- CICS_EPI_EVENT_END_TERM

The application should attempt to process events as quickly as possible, because some events describe the state of the terminal resource. If these events are not processed, the application may receive unexpected error return codes when it tries to issue EPI functions, because the state maintained inside the EPI might not match the state maintained in the application. For example, an application can start a transaction only if the terminal resource is not currently running another transaction. The **CICS_EPI_EVENT_END_TRAN** event signals the end of a transaction, and so the application and the EPI enter a state in which new transactions can be started for that terminal resource. The **CICS_EPI_EVENT_START_ATI** event indicates that an ATI transaction has started at the terminal resource, and therefore that the terminal resource has entered a state in which the starting of transactions is no longer allowed. If the application calls **CICS_EpiStartTran** after the

CICS_EPI_EVENT_START_ATI event has been notified, but before the event has been retrieved, the **CICS_EpiStartTran** call fails, even though, to the application, it may appear that it is still in a state in which it should succeed.

When a client application is driven with an event or callback, it should issue a **CICS_EpiGetEvent** to get the associated event. In certain timing conditions, the CICS_EPI_EVENT_START_ATI may already have been notified from a previous **CICS_EpiGetEvent**. The **CICS_EpiGetEvent** issued after the callback can receive CICS_EPI_ERR_NO_EVENT (if CICS_EPI_NOWAIT is specified for the **Wait** parameter) or wait until a subsequent event is received (if CICS_EPI_NOWAIT is specified for the **Wait** parameter). Note that this can happen after a CICS_EPI_EVENT_START_ATI is received.

Sending and receiving data

When a transaction sends data to a terminal resource, the EPI generates either a CICS_EPI_EVENT_SEND event or a CICS_EPI_EVENT_CONVERSE event.

The data sent might be data from the transaction, or it might be messages produced by the server, including error messages. You should consult your server documentation to see what messages are possible. An EPI application should analyze the data stream to see if an error has occurred.

The CICS_EPI_EVENT_SEND event indicates that data was sent but that no reply is required. Typically this would result from an EXEC CICS SEND command, but in some servers it would result from an EXEC CICS CONVERSE command. (In the latter case, a CICS_EPI_EVENT_CONVERSE event occurs later to tell the application to send a data stream back to the transaction in the server.)

The CICS_EPI_EVENT_CONVERSE event indicates that a reply is required, and would typically result from an EXEC CICS RECEIVE or EXEC CICS CONVERSE command. The application should respond to this event by issuing a **CICS_EpiReply** call to provide the response data. The **CICS_EpiReply** function should be issued only to respond to a CICS_EPI_EVENT_CONVERSE event; if it is issued at any other time, an error is returned.

Managing pseudoconversations

CICS transactions can operate in a pseudoconversational mode. The conversation between a terminal resource and a server is broken up into a number of segments, each of which is a separate transaction. As each transaction ends, it provides the name of the transaction to be run to process the next input from the terminal resource. (It uses EXEC CICS RETURN with the TRANSID option to do this.) The CICS_EPI_EVENT_END_TRAN event tells the application whether the transaction just ended has specified a

External presentation interface

transaction to process the next input, and which transaction has been specified. The application must not attempt to start a different transaction, but must use **CICS_EpiStartTran** to start the transaction specified by the **CICS_EPI_EVENT_END_TRAN** event.

Security in the EPI

The EPI is implemented as a set of intersystem conversations between CICS client and server; for example, a conversation is allocated for:

- Each terminal resource that is installed; refer to the **CICS_EpiAddTerminal** and **CICS_EpiAddExTerminal** functions.
- Each transaction that is started; refer to the **CICS_EpiStartTran** function.
- Each terminal resource that is deleted from the server; refer to the **CICS_EpiDelTerminal** function.

A userid and password may be required for each intersystem conversation; the requirement is determined by the way in which CICS client and CICS server have been configured.

The client application can specify the userid and password in the **CICS_EpiAttributes_t** structure passed to the **CICS_EpiAddExTerminal** function. The values are terminal specific and are passed to the server in the intersystem conversation allocated to install the terminal and in subsequent intersystem conversations relating to the installed terminal. The values can be changed at any time by the **CICS_EpiSetSecurity** function.

The client application can pass the userid and password to the **CICS_SetDefaultSecurity** function. The values are server-specific and are used whenever the terminal-specific values are not available. The values can be changed at any time by the **CICS_SetDefaultSecurity** function.

If the userid and password are required for intersystem conversations and have not been set by the **CICS_EpiAddExTerminal** or **CICS_EpiSetSecurity** or **CICS_SetDefaultSecurity** functions then the CICS client may use other methods, for example, via a pop-up window, to determine the values. (Note that pop-up windows are not supported on AIX and Solaris).

Signon incapable terminals

The userid determined by the CICS client is authenticated by the server for each transaction started at a signon incapable terminal. The transaction is executed in the server with the authorities assigned to the authenticated userid.

If no userid is passed by the CICS client, then the transaction is executed in the server with the authorities assigned to the default userid.

Signon capable terminals

The userid, if any, determined by the CICS client may be authenticated by the CICS server for each transaction started at a signon capable terminal. Such transactions are executed in the CICS server with the authorities assigned to a userid determined by the client application or to the default userid.

The first transaction(s) to be started at the terminal are executed with the authorities assigned to the default userid.

Signon transactions are CICS-supplied or user-written. In both cases, the userid and password are determined by the client application and are embedded in the 3270 data stream passed to the **CICS_EpiStartTran** or **CICS_EpiReply** functions. If the userid is authenticated then subsequent transactions started at the terminal are executed in the CICS server with the authorities assigned to the authenticated userid.

The client application may start a signoff transaction at the terminal. The userid may be signed off by the server following a predefined period of inactivity. In either case, subsequent transactions started at the terminal are executed with the authorities assigned to the default userid.

Note: A CICS server may reject an attempt to install a signon capable terminal. This would be the case for CICS for OS/400 where all authentication is managed by the operating system.

DCE protocol considerations

The operator might need DCE authorization to use the EPI application, depending on the local configuration. If the EPI program runs for a long time, for example longer than thirty minutes, you must make sure that the credential lifetime and renewable lifetime of the DCE principal are long enough to cover the maximum allowed duration of the EPI program. If an application's DCE credentials expire, all terminal connections are lost, and the next EPI call returns CICS_EPI_ERR_NOT_INIT.

Whether userids and passwords are encrypted depends on the communications protocol being used between client and server.

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 77.

EPI constants and data structures

EPI constants

The following constants are referred to symbolically in the descriptions of the EPI data structures, functions, and events in this chapter. Their values are given here to help you understand the descriptions. However, your code should always use the symbolic names of EPI constants provided for the programming language you are using.

Lengths of fields

- CICS_EPI_SYSTEM_MAX (8)
- CICS_EPI_DESCRIPTION_MAX (60)
- CICS_EPI_NETNAME_MAX (8)
- CICS_EPI_TRANSID_MAX (4)
- CICS_EPI_ABEND_MAX (4)
- CICS_EPI_DEVTTYPE_MAX (16)
- CICS_EPI_ERROR_MAX (60).
- CICS_EPI_PASSWORD_MAX (10)
- CICS_EPI_USERID_MAX (10)
- CICS_EPI_MAPNAME_MAX (7)
- CICS_EPI_MAPSETNAME_MAX (8)
- CICS_EPI_TERMID_MAX (4)

Relating to TermIndex

- CICS_EPI_TERM_INDEX_NONE 0xFFFF.

Version numbers (See “EPI versions” on page 77)

- CICS_EPI_VERSION_100
- CICS_EPI_VERSION_101.
- 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**
- **CICS_EpiSysError_t.**

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

CICS_EpiSystem_t

Purpose: The **CICS_EpiSystem_t** structure contains the name and description of a CICS server. An array of these structures is returned from the **CICS_EpiListSystems** function.

Fields:

SystemName

A string naming the CICS server. It can be passed as a parameter to the **CICS_EpiAddTerminal** and **CICS_EpiAddExTerminal** functions, to identify the CICS server in which the terminal resource should be installed. If the name is shorter than **CICS_EPI_SYSTEM_MAX** characters, it is padded with nulls to a length of **CICS_EPI_SYSTEM_MAX + 1**.

Description

A string giving a brief description of the server. If the description is shorter than **CICS_EPI_DESCRIPTION_MAX**, it is padded with nulls to a length of **CICS_EPI_DESCRIPTION_MAX + 1**.

EPI constants and data structures

CICS_EpiAttributes_t

Purpose: The **CICS_EpiAttributes_t** structure holds information about the attributes to be associated with a terminal resource installed by the **CICS_EpiAddExTerminal** function.

This data structure is only supported for CICS_EPI_VERSION_200.

Fields:

EpiAddType

An indication of whether or not the application is prepared to wait until the request to install the terminal is complete. One of the following values should be used:

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

An indication of whether or not the application may start server-provided signon and signoff transactions from the terminal resource. One of the following values should be used:

CICS_EPI_SIGNON_CAPABLE

The terminal resource is to be installed as signon capable.

CICS_EPI_SIGNON_INCAPABLE

The resource is to be installed as signon 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.

UserId

A string specifying the userid to be associated with the terminal resource. If the userid is shorter than CICS_EPI_USERID_MAX, it must be padded with nulls to a length of CICS_EPI_USERID_MAX+1.

Password

A string specifying the password to be associated with the terminal resource. If the password is shorter than CICS_EPI_PASSWORD_MAX characters, it must be padded with nulls to a length of CICS_EPI_PASSWORD_MAX+1.

EPI constants and data structures

CICS_EpiDetails_t

Purpose: The **CICS_EpiDetails_t** structure holds information about a terminal resource installed by the **CICS_EpiAddTerminal** or the **CICS_EpiAddExTerminal** function.

Fields:

NetName

A string specifying the VTAM-style netname of the terminal resource. If the name is shorter than **CICS_EPI_NETNAME_MAX** characters, it is padded with nulls to a length of **CICS_EPI_NETNAME_MAX + 1**.

NumLines

The number of rows supported by the terminal resource.

NumColumns

The number of columns supported by the terminal resource.

MaxData

The maximum size of data that can be sent to this terminal resource from a CICS transaction, and the maximum size of data that can be sent from this terminal resource to a CICS transaction by a **CICS_EpiStartTran** call or **CICS_EpiReply** call.

The maximum size may be defined in the model terminal definition specified by the **DevType** parameter on the **CICS_EpiAddTerminal** call that installed the terminal resource in the server. If the value either is not or can not be specified in the model terminal definition, a default value of 12000 is assumed.

ErrLastLine

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

ErrIntensify

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

ErrColor

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

ErrHighlight

The 3270 attribute defining the highlight value to be used to display error messages.

Highlight

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.

This field is only supported for CICS_EPI_VERSION_200.

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.

This field is only supported for CICS_EPI_VERSION_200.

SignonCapability

The signon capability assigned by the server to the terminal resource:

CICS_EPI_SIGNON_CAPABLE

if the application may start server-provided signon and signoff transactions at the terminal resource.

CICS_EPI_SIGNON_INCAPABLE

if the application may not start server-provided signon and signoff transactions at the terminal resource.

CICS_EPI_SIGNON_UNKNOWN

if the **CICS_EpiAddTerminal** function was used to add the terminal resource. (This value is also returned if the **CICS_EpiAddExTerminal** function was used to add the terminal resource and prerequisite changes have not been applied to the server.)

This field is only supported for CICS_EPI_VERSION_200.

EPI constants and data structures

CICS_EpiEventData_t

Purpose: The **CICS_EpiEventData_t** structure holds details of a terminal-related event. Not all fields are valid for all events, and fields that are not valid are set to nulls. This structure is an output from **CICS_EpiGetEvent**.

Fields:

TermIndex

The terminal index for the terminal resource against which this event occurred.

Event The event indicator; that is, one of the event codes listed in “EPI events” on page 114.

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 Universal Clients Version 3.1, this field was called **AbendCode**.

Data A pointer to a buffer that is updated with any terminal data stream associated with the event.

On input the **Data** parameter should be set to point to a **CICS_EpiDetails_t** structure on the first invocation of **CICS_EpiGetEvent** for a terminal being added asynchronously. The details structure is updated on return from **CICS_EpiGetEvent**.

Size The maximum size of the buffer addressed by **Data**. On return from the **CICS_EpiGetEvent** call, this contains the actual length of data returned.

EndReturnCode

The **CICS_EPI_returncode**.

This field is only supported for **CICS_EPI_VERSION_200**.

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.

This field is only supported for CICS_EPI_VERSION_200.

MapSetName

A string specifying the name of the mapset that was most recently referenced in the MAPSET option of a SEND MAP command processed for the terminal resource, if the event is a CICS_EPI_EVENT_SEND or a CICS_EPI_EVENT_CONVERSE event. If the MAPSET option was not specified on the most recent request, BMS used the map name as the mapset name. In both cases, the mapset name used may have been suffixed by a terminal suffix. If the terminal resource is not supported by BMS, or the server has no record of any mapset being sent, the value returned is spaces. If the name is shorter than CICS_EPI_MAPSETNAME_MAX characters, it is padded with spaces to this length, followed by a single null character.

This field is only supported for CICS_EPI_VERSION_200.

Note: The **Data** and **Size** fields should be set before the call to **CICS_EpiGetEvent** is made.

EPI constants and data structures

CICS_EpiSysError_t

Purpose: The `CICS_EpiSysError_t` structure holds system error information. It is an output of `CICS_EpiGetSysError`.

This data structure is not supported for `CICS_EPI_VERSION_200`.

All error information is written to the client log file.

Fields:

Cause A value, specific to the operating environment, indicating the cause of the last error.

Value A value, specific to the operating environment, indicating the nature of the last error.

Msg A text message, specific to the operating environment, describing the last error. If the message is shorter than `CICS_EPI_ERROR_MAX`, it is padded with nulls to a length of `CICS_EPI_ERROR_MAX + 1`.

EPI versions

The following descriptions of EPI functions embrace three versions of the EPI. You should consult the documentation for your client or server environment to determine which versions of the EPI they provide. You establish the version of the EPI that you need by using the **Version** parameter on the **CICS_EpiInitialize** function.

The function described in the following sections is the function for versions up to **CICS_EPI_VERSION_200**. The following functions and data structures are only supported for **CICS_EPI_VERSION_200**:

- **CICS_EpiAddExTerminal**
- **CICS_EpiPurgeTerminal**
- **CICS_EpiSetSecurity**
- **CICS_EpiAttributes_t**
- The **MapName**, **MapSetName**, **System**, **TermID**, and **SignonCapability** fields in the **CICS_EpiDetails_t** data structure

The following are not supported for **CICS_EPI_VERSION_200**:

- **CICS_EpiGetSysError**
- **CICS_EpiSysError_t**

In addition, if you initialize the EPI with **CICS_EPI_VERSION_100**, the following restrictions apply:

- The **CICS_EpiInquireSystem** function is not supported.
- The return codes **EPI_ERR_IN_CALLBACK**, **EPI_ERR_NULL_PARM**, and **EPI_ERR_SECURITY** are not supported. **EPI_ERR_FAILED** will be returned instead.
- The **System** parameter of **CICS_EpiAddTerminal** must specify a non-null string, as the mechanism for finding a default server is not supported. (If a null string is specified, **CICS_EPI_ERR_SYSTEM** is returned.)

EPI functions

This section describes the functions provided by the EPI that can be called from an application program:

- **CICS_EpiInitialize**
- **CICS_EpiTerminate**
- **CICS_EpiListSystems**
- **CICS_EpiAddTerminal**
- **CICS_EpiAddExTerminal**
- **CICS_EpiInquireSystem**

EPI functions

- **CICS_EpiDelTerminal**
- **CICS_EpiPurgeTerminal**
- **CICS_EpiSetSecurity**
- **CICS_EpiStartTran**
- **CICS_EpiReply**
- **CICS_EpiATISState**
- **CICS_EpiSenseCode**
- **CICS_EpiGetEvent**
- **CICS_EpiGetSysError.**

Table 3 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 for CICS Universal Client for OS/2 and CICS Universal Client for Windows:

COBOL \copybook\cicsepi.cbl
C \include\cics_epi.h
PL/I \plihdr\cics_epi.inc

and in the following files for CICS Universal Client for AIX and CICS Universal Client for Solaris:

COBOL /include/CICSEPI
C /include/cics_epi.h

Table 3. 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 3. Summary of EPI functions (continued)

Function name	Parameters	Return codes: CICS_EPI_
CICS_EpiAddTerminal	NameSpace System Netname DevType NotifyFn Details TermIndex	ERR_FAILED ERR_MAX_TERMS ERR_NOT_INIT ERR_SYSTEM ERR_SECURITY ERR_NULL_PARM ERR_IN_CALLBACK ERR_SERVER_DOWN NORMAL For CICS_EPI_VERSION_200 only: 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
CICS_EpiAddExTerminal	System Netname DevType NotifyFn Details TermIndex Attributes	ERR_FAILED ERR_NOT_INIT ERR_SYSTEM ERR_SECURITY 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

EPI functions

Table 3. Summary of EPI functions (continued)

Function name	Parameters	Return codes: CICS_EPI_
CICS_EpiInquireSystem	TermIndex System	ERR_BAD_INDEX ERR_FAILED ERR_NOT_INIT ERR_NULL_PARM ERR_IN_CALLBACK NORMAL
CICS_EpiDelTerminal	TermIndex	ERR_BAD_INDEX ERR_FAILED ERR_NOT_INIT ERR_TRAN_ACTIVE ERR_IN_CALLBACK NORMAL
CICS_EpiPurgeTerminal	TermIndex	ERR_BAD_INDEX ERR_FAILED ERR_NOT_INIT ERR_IN_CALLBACK ERR_VERSION NORMAL
CICS_EpiSetSecurity	TermIndex UserId Password	ERR_NOT_INIT ERR_BAD_INDEX ERR_IN_CALLBACK ERR_SYSTEM_ERROR ERR_VERSION ERR_PASSWORD_INVALID ERR_USERID_INVALID ERR_NULL_PASSWORD ERR_NULL_USERID NORMAL
CICS_EpiStartTran	TermIndex TransId Data Size	ERR_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 (v 200 only) ERR_MAX_SESSIONS (v 200 only) NORMAL

Table 3. Summary of EPI functions (continued)

Function name	Parameters	Return codes: CICS_EPI_
CICS_EpiReply	TermIndex Data Size	ERR_BAD_INDEX ERR_FAILED ERR_NO_CONVERSE ERR_NO_DATA ERR_NOT_INIT ERR_IN_CALLBACK ERR_ABENDED (v 200 only) ERR_SERVER_DOWN NORMAL
CICS_EpiATISState	TermIndex ATISState	ERR_ATI_STATE ERR_BAD_INDEX ERR_FAILED ERR_NOT_INIT ERR_IN_CALLBACK ERR_NULL_PARAM NORMAL
CICS_EpiSenseCode	TermIndex SenseCode	ERR_BAD_INDEX ERR_FAILED ERR_NOT_INIT ERR_SENSE_CODE ERR_IN_CALLBACK ERR_VERSION NORMAL
CICS_EpiGetEvent	TermIndex Wait	ERR_BAD_INDEX ERR_FAILED ERR_MORE_DATA ERR_MORE_EVENTS ERR_NO_EVENT ERR_NOT_INIT ERR_WAIT ERR_NULL_PARAM ERR_IN_CALLBACK NORMAL
CICS_GetSysError	TermIndex SysErr	ERR_NOT_INIT ERR_BAD_INDEX ERR_FAILED ERR_NULL_PARAM ERR_VERSION NORMAL

Refer to the definitions of the functions to discover the types and usage of the parameters, the data structures used by the functions, and the meanings of the return codes.

EPI functions

CICS_EpiInitialize

CICS_EpiInitialize	Version
--------------------	---------

Purpose

The **CICS_EpiInitialize** function initializes the EPI. All other EPI calls from this application are invalid before this call is made.

Microsoft Windows 3.1 only: See “Microsoft Windows 3.1 considerations” on page 125 for further details of the use of this function in a Microsoft Windows 3.1 environment.

Parameters

Version

The version of the EPI for which this application is coded. This makes it possible for old applications to remain compatible with future versions of the EPI. The version described here is CICS_EPI_VERSION_200. (See “EPI versions” on page 77 for more information.)

The EPI uses this parameter only for input.

Return codes

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_IS_INIT

The EPI is already initialized.

CICS_EPI_ERR_VERSION

The EPI cannot support the version requested.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiTerminate

CICS_EpiTerminate

Purpose

The **CICS_EpiTerminate** function ends the application's use of the EPI, typically just before the application terminates. All other EPI calls (except for **CICS_EpiInitialize**) are invalid when this call has completed.

The application should issue **CICS_EpiDelTerminal** calls before terminating, to ensure that any terminal resources are deleted.

Parameters

None.

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_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NORMAL

The function completed successfully.

EPI functions

CICS_EpiListSystems

CICS_EpiListSystems	Namespace Systems List
---------------------	------------------------------

Purpose

The **CICS_EpiListSystems** function returns a list of CICS servers that are candidates to act as servers for EPI requests. There is no guarantee that a communications link exists between the client 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 69 for the contents of the structure.

EPI applications should call this function immediately after each **CICS_EpiInitialize** call made to determine which CICS servers are available.

Parameters

Namespace

A pointer reserved for future use. Ensure that this is a null pointer.

Systems

A pointer to a number. On entry to the function, this number specifies the number of elements in the array specified in the **List** parameter. This value should accurately reflect the amount of storage that is available to the EPI to store the result. On return, it contains the actual number of servers found.

The EPI uses this parameter for both input and output.

List An array of **CICS_EpiSystem_t** structures that are filled in and returned by the function. The application should provide the storage for the array and must set the **Systems** parameter to indicate the number of elements in the array. The first name in the list is the default server. However, the way in which the default is defined is operating system dependent.

The EPI uses this parameter only for output.

Return codes

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_MORE_SYSTEMS

There was not enough space in the **List** array to store the details of all

the CICS servers found. The supplied array has been filled, and the **Systems** parameter has been updated to contain the total number of servers found, thus allowing you to reallocate an array of suitable size and try the function again.

CICS_EPI_ERR_NO_SYSTEMS

No CICS servers can be located. In this case, the value returned in **Systems** is zero.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_NULL_PARM

Systems is a null pointer.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NORMAL

The function completed successfully. The number of systems found is at least one, and does not exceed the value supplied as input in the **Systems** parameter.

EPI functions

CICS_EpiAddTerminal

CICS_EpiAddTerminal	NameSpace
	System
	NetName
	DevType
	NotifyFn
	Details
	TermIndex

Purpose

The **CICS_EpiAddTerminal** function installs a new terminal resource, or reserves an existing terminal resource, for the application's use. It provides a terminal index, which should be used to identify the terminal resource on all further EPI calls. It also provides the information defined in the **CICS_EpiDetails_t** data structure.

There is a limit on the number of terminals you can add with this operation: The maximum varies according to the resources available on the client system.

Microsoft Windows 3.1 only: See "Microsoft Windows 3.1 considerations" on page 125 for further details of the use of this function in a Microsoft Windows 3.1 environment.

Note: The **CICS_EpiAddTerminal** function adds terminal resources whose signon capability is dependant upon the server in which the terminal resource is installed, For example, they would be signon incapable on CICS Transaction Server for OS/390 servers.

Parameters

NameSpace

A pointer reserved for future use. Ensure that this is a null pointer.

System

A pointer to a null-terminated string that specifies the name of the server in which the terminal resource is to be installed or reserved. If the name is shorter than **CICS_EPI_SYSTEM_MAX** characters, it must be padded with nulls to a length of **CICS_EPI_SYSTEM_MAX + 1**.

If the string is all nulls, then a server, currently the default server, is selected by the EPI. To determine the name of the server chosen, use **CICS_EpiInquireSystem**. (Specifying a null string is allowed only if the EPI was initialized with **EPI_VERSION_101**, or greater.)

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, so these should be used with care when communicating with such servers.

The use of **NetName** is as follows:

1. If a name is supplied using the **NetName**, and it matches the name of an existing terminal resource in the server, the server attempts to reserve that terminal resource.
2. If a name is supplied, but it does not match the name of an existing terminal resource in the server, the server installs a terminal resource using the model terminal definition specified by the **DevType** parameter described below, and gives it the input name. (If **DevType** is a null pointer, `CICS_EPI_ERR_FAILED` is returned.)
3. If **NetName** is a null pointer, then a terminal resource is installed using the model terminal definition specified in **DevType**. If **DevType** is a null pointer, the selected terminal type is not predictable, so you are advised to use **DevType** to ensure consistent results. The name of the terminal resource is returned in the **NetName** field of the `CICS_EpiDetails_t` structure.

The EPI uses this parameter only for input.

DevType

A pointer to a null-terminated string that is used in the server to select a model terminal definition from which a terminal resource definition is generated, or a null pointer.

If a string is supplied that is shorter than `CICS_EPI_DEVTYPE_MAX` characters, it should be padded with nulls to a length of `CICS_EPI_DEVTYPE_MAX + 1`.

EPI functions

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, so these should be used with care when communicating with such servers.

The EPI uses this parameter only for input.

NotifyFn

A pointer to a callback routine that is called whenever an event occurs for the terminal resource, such as the arrival of an ATI request. If a callback routine is not required, this parameter should be set to null.

The EPI uses this parameter only for input.

DOS and Microsoft Windows 3.1: The function is ignored and never called even if specified.

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_MAX_TERMS

The maximum number of terminal resources supported by the EPI for this process has been reached.

This return code applies for Windows 3.1 only.

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.

This is supported for CICS_EPI_VERSION_200 only.

CICS_EPI_ERR_MODELID_INVALID

The function failed because an invalid Model terminal definition was supplied.

This is supported for CICS_EPI_VERSION_200 only.

CICS_EPI_ERR_NOT_3270_DEVICE

The function failed because the device type supplied was not for a 3270 device.

This is supported for CICS_EPI_VERSION_200 only.

CICS_EPI_ERR_ALREADY_INSTALLED

The function failed because the terminal was already installed.

This is supported for CICS_EPI_VERSION_200 only.

CICS_EPI_ERR_CCsid_INVALID

The function failed because an invalid CCSID was supplied.

This is supported for CICS_EPI_VERSION_200 only.

CICS_EPI_ERR_SERVER_BUSY

The function failed because the server was busy.

This is supported for CICS_EPI_VERSION_200 only.

CICS_EPI_ERR_RESOURCE_SHORTAGE

The CICS server or client did not have enough resources to complete the terminal install.

EPI functions

CICS_EPI_ERR_MAX_SESSIONS

There were not enough communication resources to satisfy this request.

CICS_EPI_ERR_MAX_SYSTEMS

An attempt was made to start connections to more servers than your configuration allows.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiAddExTerminal

CICS_EpiAddExTerminal	System
	NetName
	DevType
	NotifyFn
	Details
	TermIndex
	Attributes

Purpose

Supported only for EPI Version 2 or later.

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 should be used to identify the terminal resource on all further EPI calls. It also provides the information defined in the **CICS_EpiDetails_t** data structure.

Some attributes, for example the character set and encoding scheme to be used for 3270 data and the signon capability, may be determined by the application. These attributes are specified in the **CCSID** and **SignonCapability** fields in the **CICS_EpiAttributes_t** structure.

Parameters**System**

A pointer to a null-terminated string that specifies the name of the server in which the terminal resource is to be installed or reserved. If the name is shorter than **CICS_EPI_SYSTEM_MAX** characters, it must be padded with nulls to a length of **CICS_EPI_SYSTEM_MAX + 1**.

If the string is all nulls, then a server, currently the default server, is selected by the EPI. To determine the name of the server chosen, use **CICS_EpiInquireSystem**.

The EPI uses this parameter only for input.

NetName

A pointer to a null-terminated string that specifies the name of the terminal resource to be installed or reserved, or null. The interpretation of this name is server-dependent.

If a string is supplied that is shorter than **CICS_EPI_NETNAME_MAX**, it must be padded with nulls to a length of **CICS_EPI_NETNAME_MAX + 1**.

The string is transmitted to the server without conversion to uppercase.

EPI functions

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, so these should be used with care when communicating with such servers.

The use of **NetName** is as follows:

1. If a name is supplied using the **NetName**, and it matches the name of an existing terminal resource in the server, the server attempts to reserve that terminal resource.
2. If a name is supplied, but it does not match the name of an existing terminal resource in the server, the server installs a terminal resource using the model terminal definition specified by the **DevType** parameter described below, and gives it the input name. (If **DevType** is a null pointer, CICS_EPI_ERR_FAILED is returned.)
3. If **NetName** is a null pointer, then a terminal resource is installed using the model terminal definition specified in **DevType**. If **DevType** is a null pointer, the selected terminal type is not predictable, so you are advised to use **DevType** to ensure consistent results. The name of the terminal resource is returned in the **NetName** field of the **CICS_EpiDetails_t** structure.

The EPI uses this parameter only for input.

DevType

A pointer to a null-terminated string that is used in the server to select a model terminal definition from which a terminal resource definition is generated, or a null pointer.

If a string is supplied that is shorter than CICS_EPI_DEVTYPE_MAX characters, it should be padded with nulls to a length of CICS_EPI_DEVTYPE_MAX + 1.

The string is transmitted to the server without conversion to uppercase.

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

The EPI uses this parameter only for input.

NotifyFn

A pointer to a callback routine that is called whenever an event occurs for the terminal resource, such as the arrival of an ATI request. If a callback routine is not required, this parameter should be set to null.

The EPI uses this parameter only for input.

Details

A pointer to the **CICS_EpiDetails_t** structure that on return contains various details about the terminal resource that was installed or reserved. For asynchronous calls, the **Details** parameter should be set to NULL. If the pointer is not set to nulls, the details are added to the structure when the request to install the terminal resource has completed. For asynchronous calls this is done when the CICS_EPI_EVENT_ADD_TERM event occurs.

The EPI uses the fields in this structure only for output.

TermIndex

A pointer to a terminal index for the terminal resource just installed or reserved. The returned terminal index must be used as input to all further EPI function calls to identify the terminal resource to which the function is directed. The terminal index supplied is the first available integer starting from 0.

The EPI uses this parameter only for output.

Attributes

A pointer to the **CICS_EpiAttributes_t** structure that specifies attributes definable by the client application for the terminal resource that is to be installed *The structure must be set to nulls before use.*

Default attributes are assumed if the pointer is set to null.

The EPI uses this parameter only for input.

Return codes

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_SYSTEM

The specified server is not known to the client.

CICS_EPI_ERR_SECURITY

The server rejected the attempt for security reasons.

EPI functions

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 signon capable.

CICS_EPI_ERR_SERVER_DOWN

The function failed because the server was down.

CICS_EPI_ERR_PASSWORD_INVALID

The length of the password exceeds **CICS_EPI_PASSWORD_MAX**.

CICS_EPI_ERR_ADDTYPE_INVALID

The value assigned to the **EpiAddType** field in the **CICS_EpiAttributes_t** structure is neither **CICS_EPI_ADD_ASYNC** nor **CICS_EPI_ADD_SYNC**.

CICS_EPI_ERR_SIGNONCAP_INVALID

The value assigned to the **SignonCapability** field in the **CICS_EpiAttributes_t** structure is neither **CICS_EPI_SIGNON_CAPABLE** nor **CICS_EPI_SIGNON_INCAPABLE**.

CICS_EPI_ERR_USERID_INVALID

The length of the userid exceeds **CICS_EPI_USERID_MAX**.

CICS_EPI_ERR_TERMID_INVALID

The function failed because an invalid TermId was supplied.

This is supported for **CICS_EPI_VERSION_200** only.

CICS_EPI_ERR_MODELID_INVALID

The function failed because an invalid Model terminal definition was supplied.

This is supported for **CICS_EPI_VERSION_200** only.

CICS_EPI_ERR_NOT_3270_DEVICE

The function failed because the device type supplied was not for a 3270 device.

This is supported for **CICS_EPI_VERSION_200** only.

CICS_EPI_ERR_ALREADY_INSTALLED

The function failed because the terminal was already installed.

This is supported for **CICS_EPI_VERSION_200** only.

CICS_EPI_ERR_CCSID_INVALID

The function failed because an invalid CCSID was supplied.

This is supported for CICS_EPI_VERSION_200 only.

CICS_EPI_ERR_SERVER_BUSY

The function failed because the server was busy.

This is supported for CICS_EPI_VERSION_200 only.

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 client did not have enough resources to complete the terminal install.

CICS_EPI_ERR_MAX_SESSIONS

There were not enough communication resources to satisfy this request.

CICS_EPI_ERR_MAX_SYSTEMS

An attempt was made to start connections to more servers than your configuration allows.

CICS_EPI_NORMAL

The function completed successfully.

EPI functions

CICS_EpiInquireSystem

CICS_EpiInquireSystem	TermIndex System
-----------------------	---------------------

Purpose

The **CICS_EpiInquireSystem** function returns the name of the server on which a given terminal resource (identified by its terminal index) is installed.

Parameters

TermIndex

The terminal index of the terminal resource whose location is to be determined.

The EPI uses this parameter only for input.

System

A pointer to a string of length CICS_ECI_SYSTEM_MAX + 1 in which the name of the server will be returned.

The EPI uses this parameter only for output.

Return codes

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_NULL_PARM

System was a null pointer.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NORMAL

The function completed successfully. The name of the server is returned in the **System** parameter padded with nulls to a length of CICS_EPI_SYSTEM_MAX + 1.

CICS_EpiDelTerminal

CICS_EpiDelTerminal	TermIndex
----------------------------	------------------

Purpose

The **CICS_EpiDelTerminal** function deletes a previously added terminal resource. The application should not consider the deletion complete until it receives the corresponding **CICS_EPI_EVENT_END_TERM** event. The terminal index remains allocated until a **CICS_EpiGetEvent** call retrieves the **CICS_EPI_EVENT_END_TERM** event. A call to this function fails if the terminal resource is currently running a transaction. To ensure that a terminal resource is deleted, the application must wait until the current transaction finishes and process all outstanding events before issuing the **CICS_EpiDelTerminal** call.

If the terminal resource was autoinstalled, its definition is deleted from the server. When a **CICS_EpiDelTerminal** call has completed successfully for a terminal resource, use of the terminal index is restricted to **CICS_EpiGetEvent** and **CICS_EpiGetSysError** 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.

EPI functions

CICS_EpiPurgeTerminal

CICS_EpiPurgeTerminal	TermIndex
-----------------------	-----------

Purpose

Supported only for EPI Version 2 or later.

The **CICS_EpiPurgeTerminal** function purges a previously added terminal resource. The application should not consider the deletion complete until it receives the corresponding CICS_EPI_EVENT_END_TERM event.

The **CICS_EpiPurgeTerminal** call differs from the **CICS_EpiDelTerminal** call in that the application does not have to wait until the current transaction finishes or process all outstanding events before issuing the call.

If the terminal resource was autoinstalled, its definition is deleted from the server.

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

Purpose

CICS_EpiSetSecurity	TermIndex UserId Password
---------------------	---------------------------------

Supported only for EPI Version 2 or later.

The **CICS_EpiSetSecurity** function allows a client application to specify a userid and password to be associated with a terminal resource previously installed as signon incapable.

The **CICS_EpiSetSecurity** function may be invoked at any time; the userid and password will be used as further transactions are started for the terminal resource. A client-determined userid and password will be used if the function either has not been invoked for the terminal resource or has been invoked and has set the userid, and by implication the password, to nulls.

Note that the client application is responsible for verifying the userid and password.

Parameters

TermIndex

The terminal index of the terminal.

The EPI uses this parameter only for input.

UserId

A pointer to a null-terminated string that specifies the userid. If the userid is shorter than CICS_EPI_USERID_MAX characters, it must be padded with nulls to a length of CICS_EPI_USERID_MAX+1.

The EPI uses this parameter only for input.

Password

A pointer to a null-terminated string that specifies the password. If the password is shorter than CICS_EPI_PASSWORD_MAX characters, it must be padded with nulls to a length of CICS_EPI_PASSWORD_MAX+1.

The EPI uses this parameter only for input.

Return codes

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

EPI functions

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_ERR_SYSTEM_ERROR

An internal system error occurred.

CICS_EPI_ERR_VERSION

The function is not supported for the version at which the EPI was initialized.

CICS_EPI_ERR_NULL_PASSWORD

Password was a null pointer.

CICS_EPI_ERR_NULL_USERID

Userid was a null pointer.

CICS_EPI_ERR_PASSWORD_INVALID

The length of the password exceeds **CICS_EPI_PASSWORD_MAX**.

CICS_EPI_ERR_USERID_INVALID

The length of the userid exceeds **CICS_EPI_USERID_MAX**.

CICS_EPI_NORMAL

The function completed successfully.

CICS_EpiStartTran

Purpose

CICS_EpiStartTran	TermIndex TransId Data Size
-------------------	--------------------------------------

The **CICS_EpiStartTran** function starts a new transaction from a terminal resource, or continues a pseudoconversation.

- *Starting a new transaction*—do this after **CICS_EpiAddTerminal**, or after a CICS_EPI_EVENT_END_TRAN event indicated that the previous transaction **did not** specify a transaction to process the next input from the terminal resource.
- *Continuing a pseudoconversation*—do this after a CICS_EPI_EVENT_END_TRAN event that indicated that the previous transaction specified **did** specify a transaction to process the next input from the terminal resource.

If the call is successful, no further start requests can be issued for this terminal resource until the transaction ends; this is indicated by the CICS_EPI_EVENT_END_TRAN event.

Parameters

TermIndex

The terminal index of the terminal resource that is to run the transaction.

The EPI uses this parameter only for input.

TransId

A pointer to a string specifying the transaction to be run, or the null pointer. If a new transaction is being started, and this input is the null pointer, the name of the transaction is extracted from the data stream supplied in the **Data** parameter. If a pseudoconversation is being continued, and the pointer is not null, the string must be the name of the transaction returned in the preceding CICS_EPI_EVENT_END_TRAN event for this terminal resource. If the pointer is not null, and the string is shorter than CICS_EPI_TRANSID_MAX characters, it should be padded with spaces to this length.

The EPI uses this parameter only for input.

Data

A pointer to the 3270 data stream to be associated with the transaction. This parameter must not be a null pointer, because the data stream must contain at least an AID byte.

EPI functions

If a new transaction is being started, and the **TransId** parameter is the null pointer, the data stream must be at least 4 bytes long, must contain the name of the transaction to be started, and might contain data to be supplied to the transaction on its first EXEC CICS RECEIVE command.

If a new transaction is being started, and the **TransId** parameter is not the null pointer, the data stream might be only one byte (an AID byte), or 3 bytes (an AID byte and a cursor address), or longer than 3 bytes (an AID byte, a cursor address, and data and SBA commands). In the last case, the data is supplied to the transaction program on the first EXEC CICS RECEIVE command.

If a pseudoconversation is being continued, the data stream might be only one byte (an AID byte), or 3 bytes (an AID byte and a cursor address), or longer than 3 bytes (an AID byte, a cursor address, and data and SBA commands). In the last case the data is supplied to the transaction program on the first EXEC CICS RECEIVE command.

The details of the format of 3270 data streams for CICS are described in “3270 data streams for the EPI” on page 121.

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 client did not have enough resources to complete the terminal install.

This is supported for **CICS_EPI_VERSION_200** only.

CICS_EPI_ERR_MAX_SESSIONS

There were not enough communication resources to satisfy this request.

This is supported for **CICS_EPI_VERSION_200** only.

CICS_EPI_NORMAL

The function completed successfully.

EPI functions

CICS_EpiReply

CICS_EpiReply	TermIndex Data Size
---------------	---------------------------

Purpose

The **CICS_EpiReply** function sends data from a terminal resource to a CICS transaction. It should only be issued in response to a CICS_EPI_EVENT_CONVERSE event.

Parameters

TermIndex

The terminal index of the terminal resource from which the data is being sent.

The EPI uses this parameter only for input.

Data A pointer to the 3270 data stream to be sent to the transaction. This parameter must not be a null pointer, because the data stream must contain at least an AID byte. The data stream might be one byte (an AID byte), 3 bytes (an AID byte and a cursor address), or more than 3 bytes (an AID byte, a cursor address, and data and SBA commands). In the last case, what follows the cursor address is supplied to the transaction program on the first EXEC CICS RECEIVE command.

The length of the 3270 data stream must not exceed the value that was returned in **MaxData** in **CICS_EpiDetails_t** when the terminal resource was installed with **CICS_EpiAddTerminal**.

The EPI uses this parameter only for input.

Size The size of the data in bytes.

The EPI uses this parameter only for input.

Return codes

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NO_CONVERSE

No reply is expected by the terminal resource.

CICS_EPI_ERR_NO_DATA

No reply data was supplied.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_ERR_SERVER_DOWN

The function failed because the server was down.

CICS_EPI_ERR_ABENDED

The read timeout period has expired and the conversation has abended, but the **CICS_EPI_EVENT_END_TRAN** event has not yet been received by the application.

This is supported for **CICS_EPI_VERSION_200** only.

CICS_EPI_NORMAL

The function completed successfully.

EPI functions

CICS_EpiATISate

CICS_EpiATISate	TermIndex ATISate
-----------------	----------------------

Purpose

The **CICS_EpiATISate** function allows the calling application to query and alter the way in which ATI requests for a terminal resource are handled. If ATI requests are enabled (**CICS_EPI_ATI_ON**) and an ATI request is issued in the server, the request is started when the terminal resource becomes free. If ATI requests are held (**CICS_EPI_ATI_HOLD**), any ATI requests issued are queued, and started when ATI requests are next enabled.

The state for ATI requests after a **CICS_EpiAddTerminal** call is **CICS_EPI_ATI_HOLD**. The EPI application may change the state to **CICS_EPI_ATI_ON** when it is ready to allow ATI requests to be processed. (The server also maintains a ATI state for terminal resources, which is independent of the ATI state maintained in the EPI. Changes to the ATI state on the server do not affect the ATI status in the EPI.)

Parameters

TermIndex

The terminal index of the terminal resource whose ATI state is required.

The EPI uses this parameter only for input.

ATISate

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 **ATISate** value was provided.

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NULL_PARAM

ATISState was a null pointer.

CICS_EPI_NORMAL

The function completed successfully.

EPI functions

CICS_EpiSenseCode

CICS_EpiSenseCode	TermIndex SenseCode
--------------------------	--------------------------------------

Purpose

Not supported for EPI Version 2.

This function allows the calling application to inform the EPI of any errors in the 3270 data sent by a transaction.

The function has no effect for CICS Clients.

Parameters

TermIndex

The terminal index of the terminal resource for which the error is to be raised.

The EPI uses this parameter only for input.

SenseCode

The sense code failure reason, which can be one of the following values:

CICS_EPI_SENSE_OPCHECK

Errors were detected in the 3270 data stream.

CICS_EPI_SENSE_REJECT

Invalid 3270 commands were received.

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_SENSE_CODE

An invalid sense code was provided.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_VERSION

The function is not supported for the version at which the EPI was initialized.

CICS_EPI_NORMAL

The function completed successfully.

EPI functions

CICS_EpiGetEvent

CICS_EpiGetEvent	TermIndex
	Wait
	Event

Purpose

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

Remember that this call may be attempted only from the application, not from the callback routine.

Microsoft Windows 3.1 only: In a Microsoft Windows 3.1 environment, the application should call this function after receiving a message with **wParam** set to **CICS_EPI_WIN_EVENT**. See “Microsoft Windows 3.1 considerations” on page 125 for further details of the use of this function in a Microsoft Windows 3.1 environment, including the format of Windows messages posted by the EPI.

Parameters

TermIndex

The terminal index of the terminal resource for which to obtain an event. This can be set to the constant **CICS_EPI_TERM_INDEX_NONE** to indicate that the next event for any terminal resource used by this application is to be returned. The application can examine the **TermIndex** field in the returned **CICS_EpiEventData_t** structure to determine the terminal resource against which the event was generated.

The EPI uses this parameter for both input and output.

Wait An indication of what should happen if no event has been generated for the terminal resource. One of the following values should be used:

CICS_EPI_WAIT

Do not return until the next event occurs. (This value should not be used in the Microsoft Windows or DOS environments.)

CICS_EPI_NOWAIT

Return immediately with an error code. This option is used if the application elects to poll for events.

The EPI uses this parameter only for input.

Event A pointer to a **CICS_EpiEventData_t** structure that on return contains the details of the event that occurred. The **Data** field in the structure should be set to point to the data buffer that is updated with any terminal data stream associated with the event. The **Size** field should be set to indicate the maximum size of this buffer, and is updated to contain the actual length of data returned.

Return codes

CICS_EPI_ERR_BAD_INDEX

The **TermIndex** value is not a valid terminal index.

CICS_EPI_ERR_FAILED

The function failed for an unexpected reason.

CICS_EPI_ERR_MORE_DATA

The supplied data buffer was not large enough to contain the terminal data; the data has been truncated.

CICS_EPI_ERR_MORE_EVENTS

An event was successfully obtained, but there are more events outstanding against this terminal resource.

CICS_EPI_ERR_NO_EVENT

No events are outstanding for this terminal resource.

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has not been executed.

CICS_EPI_ERR_WAIT

The **Wait** parameter is not valid.

CICS_EPI_ERR_NULL_PARM

Event is a null pointer.

CICS_EPI_ERR_IN_CALLBACK

The function was called from a callback routine.

CICS_EPI_NORMAL

The function completed successfully, and there are no more events.

EPI functions

CICS_EpiGetSysError

CICS_EpiGetSysError	TermIndex SysErr
---------------------	---------------------

Purpose

Not supported for EPI Version 2.

The **CICS_EpiGetSysError** function obtains detailed information describing the last error that occurred when the **CICS_EPI_ERR_FAILED** return code was generated. The values returned in the **Cause** and **Value** fields in the supplied **SysErr** parameter can be used to further qualify the return code from any other EPI function. These values are environment-dependent. You might need to consult documentation for your client environment, and the documentation for the appropriate CICS server.

The **Msg** field in the supplied **SysErr** parameter may return a message, specific to the operating environment, describing the error that occurred. If no message is available, this field is set to nulls.

Note: In CICS Universal Clients Version 3.1, this function is reserved for backward compatibility. No information is returned by this function; all system errors are written to the clients error log.

Parameters

TermIndex

The terminal index of the terminal resource for which to obtain the detailed error code. This parameter may be set to the constant **CICS_EPI_TERM_INDEX_NONE**, if further error information associated with a **CICS_EpiInitialize**, **CICS_EpiTerminate**, **CICS_EpiListSystems**, or **CICS_EpiAddTerminal** call is required.

The EPI uses this parameter only for input.

SysErr

A pointer to a **CICS_EpiSysError_t** structure that on return contains the system error information.

The EPI uses the structure only for output.

Return codes

CICS_EPI_ERR_NOT_INIT

CICS_EpiInitialize has never been called. (If a **CICS_EpiInitialize** call is made and fails, **CICS_EpiGetSysError** will still succeed.)

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_NULL_PARM

SysErr is a null pointer.

CICS_EPI_VERSION

The function is not supported for the version at which the EPI was initialized.

CICS_EPI_NORMAL

The function completed successfully.

EPI events

EPI events

EPI events occur when CICS has data to pass to the EPI application. The application can handle EPI events in a variety of ways. (See “Events and callbacks” on page 63.) Whichever mechanism is used, the data from CICS is obtained by calling **CICS_EpiGetEvent**.

CICS_EPI_ADD_TERM**Purpose**

The CICS_EPI_EVENT_ADD_TERM event indicates that an asynchronous request to install a terminal resource has completed. If the terminal resource was installed details will have been placed in the **CICS_EpiDetails_t** structure, pointed to by **Data**.

Fields completed

Event The CICS_EPI_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.

EPI events

CICS_EPI_EVENT_SEND

Purpose

The CICS_EPI_EVENT_SEND event indicates that a transaction has sent some 3270 data to a terminal resource, typically as a result of an EXEC CICS SEND command. No reply is expected, and none should be attempted.

Fields completed

Event The CICS_EPI_EVENT_SEND event code.

Data A pointer to the buffer that is updated to contain the data sent by the transaction. See “3270 data streams for the EPI” on page 121 for details of the data stream format.

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

CICS_EPI_EVENT_CONVERSE

Purpose

The CICS_EPI_EVENT_CONVERSE event indicates that a transaction is expecting a reply as a result of either an EXEC CICS RECEIVE command, or an EXEC CICS CONVERSE command.

The application should issue a **CICS_EpiReply** call to return the data to CICS, as follows:

- If the transaction has issued an EXEC CICS RECEIVE command without specifying the BUFFER option, the buffer might contain data sent from the transaction, or it might be empty. If there is data to process, you should deal with it before replying. The reply should be sent when the data to be sent is available.
- If the transaction has issued an EXEC CICS RECEIVE BUFFER command, the data buffer contains the 3270 Read Buffer command and the **Size** field is set to 1. The reply should be sent immediately.

Fields completed

Event The CICS_EPI_EVENT_CONVERSE event code.

Data A pointer to the buffer that is updated to contain the data sent by the transaction, as defined above.

Size The length of the data in the buffer. This may be set to zero to indicate that no data was sent, but a reply is still expected.

EPI events

CICS_EPI_EVENT_END_TRAN

Purpose

The CICS_EPI_EVENT_END_TRAN event indicates the end of a transaction that was running against a terminal resource. If the transaction failed, the **EndReason** and **EndReturnCode** specify the cause. If the transaction completed normally, the **EndReason** field is set to CICS_EPI_TRAN_NO_ERROR and **EndReturnCode** is set to CICS_EPI_NORMAL. If the transaction was pseudoconversational, the **TransId** field contains the name of the next transaction required. The application should start this transaction by issuing a **CICS_EpiStartTran** call.

The CICS_EPI_EVENT_END_TRAN event occurs when a transaction running against a terminal resource abends or ends following execution of a RETURN command for which the IMMEDIATE option was not specified.

Fields completed

Event The CICS_EPI_EVENT_END_TRAN event code.

EndReason

An indication of what caused the end transaction event. It can be one of the following values:

CICS_EPI_TRAN_NO_ERROR

Normal transaction termination.

CICS_EPI_TRAN_NOT_STARTED

The transaction failed to start.

CICS_EPI_TRAN_STATE_UNKNOWN

The transaction failed to complete.

CICS_EPI_READTIMEOUT_EXPIRED

The read timeout expired.

TransId

The name of the next transaction to start, if the previous transaction was pseudoconversational. This name is 4 characters long and null-terminated. If there is no next transaction, the field is set to nulls.

EndReturnCode

Contains the CICS_EPI_returncode.

CICS_EPI_EVENT_START_ATI**Purpose**

The CICS_EPI_EVENT_START_ATI event indicates that an ATI transaction has been started against the terminal resource. If the terminal resource receives an ATI request while it is running another transaction, the request is held until the transaction ends. The transaction is then started on behalf of the terminal resource, and the CICS_EPI_EVENT_START_ATI event is generated to inform the application.

Fields completed

Event The CICS_EPI_EVENT_START_ATI event code.

TransId

The name of the transaction that was started. This name is 4 characters long and null-terminated.

EPI events

CICS_EPI_EVENT_END_TERM

Purpose

The CICS_EPI_EVENT_END_TERM event indicates that a terminal resource no longer exists. After this event, the terminal index that was previously used for the terminal resource is not valid. If the EPI detects that a CICS server has shut down, CICS_EPI_EVENT_END_TERM events are generated for all terminal resources that the application has installed in that server and not subsequently deleted.

Fields completed

Event The CICS_EPI_EVENT_END_TERM event code.

EndReason

An indication of why the terminal resource was deleted. It can be one of the following values:

CICS_EPI_END_SIGNOFF

The terminal resource was signed off. This can be as a result of running the CESF transaction or of calling the **CICS_EpiDelTerminal** function.

CICS_EPI_END_SHUTDOWN

The CICS server is shutting down.

CICS_EPI_END_OUTSERVICE

The terminal resource has been switched out of service.

CICS_EPI_END_UNKNOWN

An unexpected error has occurred.

CICS_EPI_END_FAILED

An attempt to delete a terminal resource failed.

3270 data streams for the EPI

The data streams implemented for the EPI follow those defined in the *3270 Data Stream Programmer's Reference*. All data flows for the EPI are in ASCII format, and structured fields are not supported. Data flows are defined under the following topics in the *3270 Data Stream Programmer's Reference*:

- Introduction to the 3270 data stream (excluding structured fields)
- 3270 data stream commands
- Character sets, orders and attributes
- Keyboard and printer operations.

The use of 3270 data streams within the EPI, and some nonstandard orders and attributes, are defined here.

The supplied C header file CICS3270.H, COBOL copybook CICS3270.CBL, and PL/I include file CICS3270.INC contain constants and conversion tables that you will find useful in handling 3270 data streams.

Note that if a CICS user transaction issues EXEC CICS SEND and EXEC CICS RECEIVE commands, CICS does not process the data (after the initial control bytes) that is passed between the EPI application and the CICS transaction. In this case, the data buffer may be used to pass user-specified data between the programs. Be aware that the contents of the data buffer may be code-page converted if the buffer is passed between CICS systems, in which case the data should be limited to ASCII and EBCDIC characters.

If a CICS transaction issues EXEC CICS SEND MAP and EXEC CICS RECEIVE MAP commands, CICS converts the data from the BMS structure to a 3270 data stream. In this case, the application receives 3270 data from CICS and should return valid 3270 data to be converted for the transaction.

The distributed CICS products (that is, CICS Universal Clients, CICS for OS/2) support the ASCII-7 subset of the 3270 data stream architecture. This precludes the use of 14- and 16- bit addresses and/or structured fields, and means that only 12-bit SBA addressing is supported. Because of this restriction, the maximum screen size for EPI terminals is 27 rows by 132 columns. The CICS/390 and CICS/400 products support the EBCDIC 3270 data stream architecture. However transactions on these servers must avoid the use of 14- and 16- bit addresses and/or structured fields if these are to be started from one of the distributed CICS products.

Inbound data streams (EPI to CICS)

AID (1 byte)	Cursor address (2 bytes)	Data buffer (variable length)
-----------------	-----------------------------	----------------------------------

EPI applications send 3270 data to CICS on calls to the following functions:

3270 data streams

- CICS_EpiStartTran
- CICS_EpiReply.

The format in both cases is the same. The data stream must be a minimum of 3 non-null bytes, representing the AID and cursor address; the sole exception to this is if the AID represents the CLEAR key or a PA key, when the data stream may consist of the AID only. These fields are passed to the CICS transaction in the EIBAID and EIBCPOSN fields of the EIB.

The contents of the data buffer consist of:

- ASCII displayable characters with embedded 3270 control characters, when it is passed to an EXEC CICS RECEIVE MAP command.
- User-specified data, when it is passed to an EXEC CICS RECEIVE command.

On starting a transaction, the transaction ID is extracted from the start of the data buffer as follows:

- If a set buffer address (SBA) order is present at the start of the data buffer, the transaction ID is extracted from the 4th through 7th bytes of the buffer.
- If an SBA is not present at the start of the data buffer, the transaction ID is extracted from the 1st through 4th bytes of the buffer.

In either case, the transaction ID may be shorter than 4 bytes, being delimited by either another SBA, an ASCII space, or the end of the string.

The contents of the data buffer passed on the start of a CICS transaction are available to the transaction in response to an initial EXEC CICS RECEIVE command.

When the application replies, the contents of the data buffer are available in an unconverted form in response to an EXEC CICS RECEIVE command or converted to a BMS structure in response to an EXEC CICS RECEIVE MAP command.

Note: It is the EPI programmer's responsibility in the latter case to ensure that the data is formatted correctly so that the conversion succeeds.

Outbound data streams (CICS to EPI)

Command (1 byte)	Write control character (1 byte)	Data buffer (variable length)
---------------------	--	----------------------------------

The 3270 commands are either write or read commands, instructing the EPI to process the data or to reply with data respectively.

On a CICS_EPI_EVENT_SEND event, the command is one of the following 3270 write commands:

- Write
- Erase/Write
- Erase/Write Alternate
- Erase All Unprotected.

The first three commands are followed by a write control character (WCC) and data. An Erase All Unprotected command has neither WCC nor data. The Write Structured Field command is not generated by CICS and is therefore not supported for the EPI.

The contents of the data buffer consist of:

- ASCII displayable characters with embedded 3270 control characters, when it is passed from an EXEC CICS SEND MAP command.
- User-specified data, when it is passed from an EXEC CICS SEND command.

A CICS_EPI_EVENT_CONVERSE event specifies a read command. The contents of the data stream vary with the source of the event, as follows:

- If the event is the result of an EXEC CICS RECEIVE command, the data buffer might contain data sent by the transaction, or it might be empty. The EPI program should reply when the data to be sent is available.
- If the event is the result of an EXEC CICS RECEIVE BUFFER command, the data buffer contains the 3270 Read Buffer command. This should be processed as described in the *3270 Data Stream Programmer's Reference*.

The Read Modified and Read Modified All commands are not generated by CICS and are therefore not supported for the EPI.

3270 order codes

3270 orders are included in both inbound and outbound data streams to provide additional control function. Table 4 on page 124 lists the order codes that may occur in 3270 data streams, and shows whether they relate to inbound or outbound data streams, or both.

3270 data streams

Table 4. Order codes occurring in 3270 data streams

Order code	Inbound	Outbound
Start field (SF)	Yes	Yes
Start field extended (SFE)	Yes	Yes
Set buffer address (SBA)	Yes	Yes
Set attribute (SA)	Yes	Yes
Modify field (MF)	No	Yes
Insert cursor (IC)	No	Yes
Program tab (TB)	No	Yes
Repeat to address (RA)	No	Yes
Erase unprotected to address (EUA)	No	Yes
Graphic escape (GE)	No	No

Note: The *3270 Data Stream Programmer's Reference* states that the SFE, SA, and MF orders are not supported in ASCII. However, they do occur in 3270 data streams for the EPI, where they take the following values:

SFE	X'10'
SA	X'1F'
MF	X'1A'

MF was formerly assigned the value X'1E'. However the Field Mark format control order is assigned the same value. Without the change to X'1A', an EPI application would be unable to distinguish between MF and FM.

Each of these orders is followed by one or more attribute type-value pairs. The count of attribute pairs and the attribute type are both binary values, and are thus as defined in the *3270 Data Stream Programmer's Reference*. However, the contents of the attribute value field may vary from those defined in the *3270 Data Stream Programmer's Reference* as follows:

- If the attribute type is less than or equal to X'C0' (for example, a color), the attribute value is defined as an EBCDIC value in the *3270 Data Stream Programmer's Reference*. The EPI uses the ASCII equivalent of the EBCDIC value; for example, red is defined as X'F2' in the *3270 Data Stream Programmer's Reference*, and should be defined as X'32' in the EPI data stream.
- If the attribute type is greater than X'C0' (for example, field outlining), the attribute value is a binary value. The EPI uses the values defined in the *3270 Data Stream Programmer's Reference*.

Further details of 3270 orders and other control characters are supplied in the files named in the following table.

	Supplied file
COBOL copybook	CICS3270.CBL
C header file	CICS3270.H
PL/I include file	CICS3270.INC

Microsoft Windows 3.1 considerations

Note: The following information applies only to the IBM CICS Client for Windows.

The **CICS_EpiInitialize** function has two additional parameters:

hWnd

The handle of the Microsoft Windows window to which EPI messages will be posted.

MsgId

The message identifier to be used for posting EPI messages to the window specified in the **hWnd** parameter.

The EPI application is posted a Windows message (using the **hWnd** and **MsgId** values) in any of the following situations:

- The **CICS_EpiInitialize** call has completed.
- The **CICS_EpiAddTerminal** call has completed.
- An incoming event for a terminal resource has been received. (**CICS_EpiGetEvent** should be called to receive the event data.)

Values are assigned to the **wParam** and **lParam** parameters within these messages as follows. In the case of **lParam**, the LOWORD of this parameter is used.

Microsoft Windows 3.1 considerations

wParam

- CICS_EPI_WIN_INITIALIZED**
CICS_EpiInitialize complete
- CICS_EPI_WIN_INSTALLED**
CICS_EpiAddTerminal complete
- CICS_EPI_WIN_EVENT**
An incoming terminal event

lParam

X'0000'
CICS_EpiInitialize has completed.

Terminal index

Index number of the terminal resource installed by a successful **CICS_EpiAddTerminal**, or of the terminal resource to which an incoming event belongs.

X'FFFF'
CICS_EpiAddTerminal has completed, but terminal resource installation failed.

After receiving a message with **wParam** set to **CICS_EPI_WIN_INITIALIZED**, the application should call **CICS_EpiGetSysError** to check for any install errors.

If **lParam** is set to **X'FFFF'**, **CICS_EpiGetSysError** should be called with the **TermIndex** parameter set to **CICS_EPI_TERM_INDEX_NONE**. (A terminal index will not have been allocated if the terminal resource install failed.)

If **lParam** is not set to **X'FFFF'**, **CICS_EpiGetSysError** should be called with the **TermIndex** parameter set to the **lParam** value.

Under Microsoft Windows, the EPI functions **CICS_EpiInitialize** and **CICS_EpiAddTerminal** are asynchronous (all other functions are synchronous). This means that the application should wait for a message to be posted before continuing after each of these two calls.

Chapter 4. Creating ECI and EPI application programs

This chapter contains language-dependent information about the external access interfaces, and is organized as follows:

“Writing the non-CICS applications”

“Making ECI calls” on page 128

“Making EPI calls” on page 132

“Compiling and linking applications” on page 133

This chapter does not deal with testing or debugging ECI and EPI applications. You should refer to the programming manuals for the environment in which you are working.

Writing the non-CICS applications

ECI and EPI applications are standard stand-alone programs that can be run on any of the client systems, and can use the server implementations of the ECI and EPI on CICS for OS/2 servers. Programs that do not make operating-system-specific calls are portable between these environments. *The sample programs and associated build files for your environment illustrate much of what is discussed here.*

An application program may use the facilities of both ECI and EPI.

An application must be constructed as a single process, though in environments in which a process can generate several threads, an application can be multi-threaded.

Applications may be written in C (or C++), COBOL, PL/I, or REXX, though not all these languages are available in every environment.

The following table shows for each programming environment

- the names of the C header file, COBOL copybook, or PL/I include file for the ECI
- the names of the C header file, COBOL copybook, or PL/I include file for the EPI
- the names of the type definition files for either interface for C and PL/I programs. (Type definition files are not used with COBOL.)

Creating applications

Use	File
ECI: C programs	CICS_ECI.H
ECI: COBOL programs	CICSECI.CBL
ECI: PL/I programs	CICS_ECI.INC
EPI: C programs	CICS_EPI.H
EPI: COBOL programs	CICSEPI.CBL
EPI: PL/I programs	CICS_EPI.INC
Type definitions: C programs	CICSTYPE.H
Type definitions: PL/I programs	CICSTYPE.INC

Note: The file extension .CBL does not apply on AIX and Solaris.

You should study the contents of the file appropriate to the language you intend to use. Each file contains the definitions of all the entry points, types, data structures, and constants needed for writing programs for the corresponding interface.

The naming conventions for the structures, fields, and constants of the interfaces in the different languages are as follows:

- PL/I and C—the names are the same as the names used earlier in this book to describe the interfaces.
- COBOL—the names are upper-case versions of the names used earlier in this book to describe the interfaces with hyphens instead of underscores.

There are a few exceptions to this convention, and these are noted where they occur.

When compiling C programs, you may need to ensure that the structures passed to the CICS external facilities are packed. If this is necessary, the C header files will contain the `#pragma pack` directive, which should not be changed.

Making ECI calls

ECI functions are called in the manner described below.

COBOL applications must ensure that the ECI function calls are statically linked at link time by using the system linkage convention.

- For IBM VisualAge® for COBOL for OS/2, you should use the default `CALLINTERFACE(System)` linkage.

- For Micro Focus Object COBOL, you should use call-convention 8 for every program call, or use the default call-convention 0 and compile using the LITLINK compiler directive.

CICS_ExternalCall

The method of calling **CICS_ExternalCall** is shown here for COBOL, C, and for PL/I. The ECI parameter block (ECI-PARMS for COBOL, ECI_PARMS for C, and ECI_PARMS for PL/I) is used for passing parameters to the ECI. The format of the call and associated declarations is as follows:

For C programs:

```

ECI_PARMS      EciBlock;
cics_sshort_t  Response;
.
.
.
Response = CICS_ExternalCall(&EciBlock);

```

For COBOL programs:

```

CALL 'CICSEXTERNALCALL'
USING BY REFERENCE ECI-PARMS
RETURNING ECI-RETURN-CODE.

```

The name 'CICS_ExternalCall' may also be used with the Micro Focus Object COBOL compiler, but it is not recommended.

For PL/I programs:

```

dcl Response fixed bin(15);
dcl EciBlock type ECI_PARMS;
.
.
.
Response = CICS_ExternalCall(EciBlock);

```

For REXX programs:

Before using **CICS_ExternalCall**, the REXX program must add the function for **CICS_ExternalCall** to its environment as follows:

```

Call RxFuncAdd 'CICS_ExternalCall', 'cclrxeci', 'RxCICS_ExternalCall'

```

Once the function has been added, the program may use it any number of times.

The format of the call is as follows:

```

retcode = CICS_ExternalCall('ECI_PARMS.')
```

Creating applications

where `ECI_PARMS` is the name chosen for the REXX stem representing the ECI parameter block. Do not omit the period at the end of the name, or the single quotation marks surrounding it.

The values for `eci_call_type`, `eci_extend_mode`, and `eci_version` in the ECI parameter block, and for `ConnectionType`, `CicsServerStatus`, and `CicsClientStatus` in the ECI status block are character literals that match the names used earlier in this book. The following sample assignment statement shows how, having chosen `ECI_PARMS` as the stem name, the `eci_call_type` field in the ECI parameter block is set to `ECI_SYNC`.

```
ECI_PARMS.eci_call_type = 'ECI_SYNC'
```

The return code values `ECI_NO_ERROR`, and so on, are defined in the file `RXECI.MSG` supplied with the samples.

Callback routines

Each callback routine has only one parameter. You should consult the sample programs for examples of writing callback routines. Callback routines cannot be used with REXX.

CICS_EciListSystems

The format of the call in the different programming languages is as follows:

For C programs:

```
#define MAX_SYS 5
cics_sshort_t   Response;
cics_ushort_t   Count;
CICS_EciSystem_t List[MAX_SYS];
.
Count = MAX_SYS;
.
Response = CICS_EciListSystems(NULL, &Count, List);
```

For COBOL programs:

The COBOL name of the system information structure is `CICS-ECISYSTEM`, which is contrary to the naming convention stated earlier.

```
01 LIST-SIZE PIC 9(4) COMP-5.

CALL 'CICSECLISTSYSTEMS'
USING
    BY REFERENCE NULL-PTR
    BY REFERENCE LIST-SIZE
    BY REFERENCE CICS-ECISYSTEM
RETURNING ECI-RETURN-CODE.
```

The name '`_CICS_EciListSystems`' may also be used with the Micro Focus Object COBOL compiler, but it is not recommended.

For PL/I programs:

```
dcl EciRetCode fixed bin(15);
dcl count fixed bin(15);
dcl list type CICS_EciSystem_t;
.
.
.
EciRetCode = CICS_EciListSystems(null(), count, list);
```

For REXX programs:

Before using `CICS_EciListSystems`, the REXX program must add the function for `CICS_EciListSystems` to its environment as follows:

```
Call RxFuncAdd 'CICS_EciListSystems', 'cclrxeci', 'RxCICS_EciListSystems'
```

Once the function has been added, the program may use it any number of times.

The format of the call is as follows:

```
retcode = CICS_EciListSystems(, 'count', 'LIST.')
```

No parameter need be supplied for **NameSpace**, `count` is the name of the REXX variable chosen for **Systems**, and `LIST` is the name chosen for the REXX stem representing **List**. Do not omit the period at the end of the name, or the single quotation marks surrounding it.

On return from the call, the names `LIST.1.SystemName`, and `LIST.1.Description`, are used to refer to the fields in the first array element, and so on.

Debugging with REXX

Special debugging facilities are provided for ECI applications using REXX.

Setting the REXX variable `DEBUG` to any non-zero value will cause debug information to be sent to a log file. The path name of the file can be specified in the REXX variable `DEBUGFILE`, and the default is the file `CCLRXECL.LOG` in the directory where `CCLRXECL.DLL` is executing. If the file exists already, debug information is appended to it.

Making EPI calls

EPI functions

EPI functions are called in a standard manner. Examples of calling the **CICS_EpiListSystems** function follow.

COBOL applications must ensure that the EPI function calls are statically linked at link time by using the system linkage convention.

- For IBM VisualAge for COBOL for OS/2, you should use the default `CALLINTERFACE(System)` linkage.
- For Micro Focus Object COBOL, you should use call-convention 8 for every program call, or use the default call-convention 0 and compile using the `LITLINK` compiler directive.

For C programs:

```
#define MAX_SYS 5
CICS_EpiSystem_t System;
cics_ushort_t Count;
cics_sshort_t Response;
.
Count = MAX_SYS;
.
Response = CICS_EpiListSystems(NULL, &Count, &System);
```

For COBOL programs:

The following may be used:

```
01 COUNT      PIC 9(4)  COMP-5.
01 NAMESPACE  POINTER.
.
.
CALL 'CICSEPILISTSYSTEMS'
  USING BY VALUE NAMESPACE
  BY REFERENCE COUNT
  BY REFERENCE CICS-EPISYSTEM
  RETURNING EPI-RETURN-CODE.
```

For PL/I programs:

The following may be used:

```
dcl System type CICS_EpiSystem_t,
     Count fixed bin(15),
     Response fixed bin(15);
.
.
.
Response = CICS_EpiListSystems(null(), Count, System);
```

For REXX programs:

EPI functions are not available in REXX.

Callback routines

Each callback routine has only one parameter. You should consult the sample programs for examples of writing callback routines.

Compiling and linking applications

This section gives some examples showing how to compile and link typical ECI and EPI applications in the various client environments. These are examples only, and may refer to specific compilers and linkers.

You should refer to the samples supplied with your environment for more information about compiling and linking programs.

IBM CICS Client for DOS Version 2

For examples of programs that call the ECI and EPI, refer to the samples supplied with the client in the following directories:

- \CICSCLI\SAMPLES\C
- \CICSCLI\SAMPLES\COBOL

For C programs:

```
cl /c /DCICS_DOS program.c

link program.obj,,,ccldos.lib;
```

Notes:

- Example compiler used is Microsoft C Optimizing Compiler Version 6.00A.
- The constant CICS_DOS must be defined to the compiler using the /DCICS_DOS option.
- Programs may use any of the standard memory models.
- The application must be linked with the CCLDOS.LIB library in addition to the standard C runtime libraries.
- Callback functions are not supported in DOS.

For COBOL programs:

```
cobol program,,,,

link program.obj,,,lcobol.lib+cobapi.lib+ccldos.lib;
```

Notes:

Creating applications

- Example compiler used is Micro Focus COBOL Version 3.0.
- Programs should use the LARGE memory model.
- ECI and EPI function calls should be linked using call-convention 8.
- The application must be linked with the CCLDOS.LIB library and must use the COBOL static link libraries.
- Callback functions are not supported in DOS or COBOL.

IBM CICS Client for Windows Version 2

For examples of programs that call the ECI and EPI, refer to the samples supplied with the client in the following directory:

- \CICSCLI\SAMPLES\C

For C programs:

```
cl /c /Gsw /DCICS_WIN program.c  
  
link program.obj,,,cclwin.lib,program.def
```

Notes:

- Example compiler used is Microsoft C/C++ Optimizing Compiler Version 7.00.
- The constant CICS_WIN must be defined to the compiler using the /DCICS_WIN option.
- The /Gsw options are recommended for compiling Windows programs.
- Programs may use any of the standard memory models.
- The application must be linked with the CCLWIN.LIB library in addition to the standard C runtime and Windows libraries.
- A STACKSIZE of at least 8K is recommended in the linker .DEF file.
- Callback functions must be declared using the CICSEXIT calling convention and must be created using the Windows function MakeProcInstance()—see samples for details.
- The Windows resource compiler will also be required to create a functional Windows program, as normal.

IBM CICS Clients for Windows NT and Windows 95 Version 2

For examples of programs that call the ECI and EPI, refer to the samples supplied with the client in the following directories:

- \CICSCLI\SAMPLES\C
- \CICSCLI\SAMPLES\COBOL

For C programs:

```
cl /c /DWIN32 /D_WIN32 /D_X86_=1 /DCICS_W32 program.c  
  
link program.obj cclwin32.lib
```

Notes:

- Example compiler used is Microsoft Visual C++ Version 2.0.
- The compiler options /DWIN32, /D_WIN32, and /D_X86_=1 are used to select the correct Windows function when processing the windows.h header file. These are standard Win32 options, and are not specific to the IBM CICS clients.
- The compiler option /DCICS_W32 must be used to define the symbol CICS_W32 to the compiler to ensure that the CICS header files are processed correctly.
- The application must be linked with the CCLWIN32.LIB library in addition to the standard C runtime and Windows libraries.
- Callback functions must be declared using the CICSEXIT calling convention—see samples for details.

For COBOL programs:

```
cobol program;  
cblnames -mPROGRAM program.obj  
link ecicobnl.obj cbllds.obj /NOD cclwin32.lib mfrts32.lib msvcr7.lib kernel32.lib
```

Notes:

- Example compiler used is Micro Focus Object COBOL Version 4.
- "program" must be replaced by the name of the program, and "PROGRAM" must be replaced by the name of the program in upper case.
- It is important to use the correct calling convention when invoking the ECI or EPI from COBOL. The sample programs use the "SPECIAL-NAMES. CALL CONVENTION 8 IS CICS." statements to achieve this.
- The application must be linked with the CCLWIN32.LIB library, in addition to the standard COBOL libraries, because a 32-bit Windows application is being generated.
- ECI or EPI callback functions are not supported in COBOL applications.

IBM CICS Client for OS/2 Version 2

For examples of programs that call the ECI and EPI, refer to the samples supplied with the client in the following directories:

- \CICSCLI\SAMPLES\C
- \CICSCLI\SAMPLES\COBOL
- \CICSCLI\SAMPLES\PLI

Creating applications

For 32-bit C programs:

```
icc /C /DCICS_OS2 program.c  
  
link386 program.obj,,,cclos232.lib,program.def
```

Notes:

- Example compiler used is IBM C/Set++.
- The constant CICS_OS2 must be defined to the compiler using the /DCICS_OS2 option.
- The application must be linked with the CCLOS232.LIB library (since it is a 32-bit application) in addition to the standard C runtime and OS/2 libraries.
- A STACKSIZE of at least 16K is recommended in the linker .DEF file.
- Callback functions must be declared using the CICSEXIT calling convention—see samples for details.

For 16-bit C programs:

```
cl /c /Gs /DCICS_OS2 program.c  
  
link program.obj,,,cclos2.lib,program.def
```

Notes:

- Example compiler used is Microsoft C Optimizing Compiler Version 6.00A.
- The constant CICS_OS2 must be defined to the compiler using the /DCICS_OS2 option.
- The /Gs option is recommended if callback functions are to be used.
- The application must be linked with the CCLOS2.LIB library (since it is a 16-bit application) in addition to the standard C runtime and OS/2 libraries.
- A STACKSIZE of at least 16K is recommended in the linker .DEF file.
- Callback functions must be declared using the CICSEXIT calling convention—see samples for details.

For 32-bit COBOL programs:

If you are using IBM VisualAge for COBOL for OS/2:

```
cob2 -c -qapost -qnosequence -qlib. prog.cb1  
  
ilink /NOFREE /NOD /NOI /MAP prog.obj,,,OS2386 iwzrlib cclos232,prog.def
```

If you are using Micro Focus Object COBOL:

```
cobol prog.cbl;  
  
cblnames -mPROG -oPROG.CBJ prog  
  
link386 /NOD /NOI prog.obj prog.cbj,,,OS2386 mfrts32 cclos232,prog.def
```

Notes:

- You must resolve ECI and EPI function calls at link time using the system linkage convention. This is the default linkage for IBM VisualAge for COBOL for OS/2. If you use Micro Focus Object COBOL, you must use call-convention 8 on all ECI and EPI functions, or use the default call-convention 0 and compile using the LITLINK compiler directive.

For 16-bit COBOL programs:

```
cobol program,,,  
  
link program.obj,,,coblib.lib+cclos2.lib,program.def
```

Notes:

- Example compiler used is Micro Focus COBOL Version 3.0.
- Programs should use the LARGE memory model.
- Programs must be compiled with the LITLINK compiler directive.
- The application must be linked with the CCLOS2.LIB library (since it is a 16-bit application) and may use either the COBOL static or dynamic link libraries.
- A STACKSIZE of at least 16K is recommended in the linker .DEF file.
- Callback functions are not supported in COBOL.

For PL/I programs:

```
pli program  
  
link386 program.obj,,,cclos232.lib,program.def
```

Notes:

- Example compiler used is IBM PL/I for OS/2 Version 1 Release 1.
- The application must be linked with the CCLOS232.LIB library (since it is a 32-bit application) in addition to the standard PL/I runtime and OS/2 libraries.
- A STACKSIZE of at least 16K is recommended in the linker .DEF file.
- Callback functions must be declared using the CICSEXIT calling convention—see samples for details.

IBM CICS Universal Clients for Windows NT and Windows 98 Version 3

For examples of programs that call the ECI and EPI, refer to the samples supplied with the client in the following directories:

- \CICS Universal Client\SAMPLES\C
- \CICS Universal Client\SAMPLES\COBOL

For C programs:

```
c1 /c /DWIN32 /D_WIN32 /D_X86_=1 /DCICS_W32 program.c  
  
link program.obj cclwin32.lib
```

Notes:

- Example compiler used is Microsoft Visual C++ Version 2.0, and IBM VisualAge for C++ Version 3.5
- The compiler options /DWIN32, /D_WIN32, and /D_X86_=1 are used to select the correct Windows function when processing the windows.h header file. These are standard Win32 options, and are not specific to the IBM CICS clients.
- The compiler option /DCICS_W32 must be used to define the symbol CICS_W32 to the compiler to ensure that the CICS header files are processed correctly.
- The application must be linked with the CCLWIN32.LIB library in addition to the standard C runtime and Windows libraries.
- Callback functions must be declared using the CICSEXIT calling convention—see samples for details.

For COBOL programs:

```
cobol program;  
cblnames -mPROGRAM program.obj  
link ecicobl.obj cbllds.obj /NOD cclwin32.lib mfrts32.lib msvcrt.lib kernel32.lib
```

Notes:

- Example compiler used is Micro Focus Object COBOL Version 4.
- "program" must be replaced by the name of the program, and "PROGRAM" must be replaced by the name of the program in upper case.
- It is important to use the correct calling convention when invoking the ECI or EPI from COBOL. The sample programs use the "SPECIAL-NAMES. CALL CONVENTION 8 IS CICS." statements to achieve this.
- The application must be linked with the CCLWIN32.LIB library, in addition to the standard COBOL libraries, because a 32-bit Windows application is being generated.
- ECI or EPI callback functions are not supported in COBOL applications.

IBM CICS Universal Client for OS/2 Version 3

For examples of programs that call the ECI and EPI, refer to the samples supplied with the client in the following directories:

- \CICS Universal Client\SAMPLES\C
- \CICS Universal Client\SAMPLES\COBOL
- \CICS Universal Client\SAMPLES\PLI

For C programs:

```
icc /C /DCICS_OS2 program.c

link386 program.obj,,,cclos232.lib,program.def
```

Notes:

- Example compiler used is IBM C/Set++.
- The constant CICS_OS2 must be defined to the compiler using the /DCICS_OS2 option.
- The application must be linked with the CCLOS232.LIB library (since it is a 32-bit application) in addition to the standard C runtime and OS/2 libraries.
- A STACKSIZE of at least 16K is recommended in the linker .DEF file.
- Callback functions must be declared using the CICSEXIT calling convention—see samples for details.

For COBOL programs:

If you are using IBM VisualAge for COBOL for OS/2:

```
cob2 -c -qapost -qnosequence -qlib. prog.cbl

ilink /NOFREE /NOD /NOI /MAP prog.obj,,,OS2386 iwzrlib cclos232,prog.def
```

If you are using Micro Focus Object COBOL:

```
cobol prog.cbl;

cblnames -mPROG -oPROG.CBJ prog

link386 /NOD /NOI prog.obj prog.cbj,,,OS2386 mfrts32 cclos232,prog.def
```

Notes:

- You must resolve ECI and EPI function calls at link time using the system linkage convention. This is the default linkage for IBM VisualAge for COBOL for OS/2. If you use Micro Focus Object COBOL, you must use call-convention 8 on all ECI and EPI functions, or use the default call-convention 0 and compile using the LITLINK compiler directive.

Creating applications

For PL/I programs:

```
pli program
```

```
link386 program.obj,,,cclos232.lib,program.def
```

Notes:

- Example compiler used is IBM PL/I for OS/2 Version 1 Release 1.
- The application must be linked with the CCLOS232.LIB library (since it is a 32-bit application) in addition to the standard PL/I runtime and OS/2 libraries.
- A STACKSIZE of at least 16K is recommended in the linker .DEF file.
- Callback functions must be declared using the CICSEXIT calling convention—see samples for details.

IBM CICS Universal Client for AIX Version 3

For examples of programs that call the ECI and EPI, refer to the samples supplied with the client in the following directories:

- /usr/lpp/cicscli/samples/c
- /usr/lpp/cicscli/samples/cobol

For C programs:

```
cc_r -c -DCICS_AIX -I/usr/lpp/cicscli/include program.c  
cc_r -o program program.o -lthreads -lc_r -lcclaix
```

Notes:

- Example compiler used is IBM C Set++ for AIX Version 3.1.4.
- The constant CICS_AIX must be defined to the compiler using the -DCICS_AIX option.
- The application must be linked with the standard AIX libpthread.a and libc_r.a libraries, as well as the libcclaix.a library.

For COBOL programs:

```
cob2_r -c -qLIB -I/usr/lpp/cicscli/include program.cbl  
xlc_r -o program program.o -lthreads -lcob2_r -lcclaix
```

Notes:

- Example compiler used is IBM COBOL Set for AIX Compiler Version 1.1.
- The application must be linked with the standard AIX libpthread.a and libcob2_r.a libraries, as well as the libcclaix.a library.

IBM CICS Universal Client for Solaris Version 3

For examples of programs that call the ECI and EPI, refer to the samples supplied with the client in the following directories:

- /opt/cicscli/samples/c
- /opt/cicscli/samples/cobol

For C programs:

```
cc -c -DCICS_SOL -I/opt/cicscli/include program.c
cc -o program program.o -lpthread -lc -lcclsol
```

Notes:

- Example compiler used is Sun Workshop Compiler C/C++ Version 4.2
- The constant CICS_SOL must be defined to the compiler using the -DCICS_SOL option.
- The application must be linked with the standard Solaris libpthread.so and libc.so libraries, as well as the libcclsol.so library.

For COBOL programs:

```
cob -x program.cbl -o program -lcclsol -lpthread
```

Notes:

- Example compiler used is Micro Focus COBOL for Solaris Version 4.2.
- The application must be linked with the standard Solaris libpthread.so library, as well as the libcclsol.so library.
- It may be necessary to set the environment variable COBCPY to pick up the correct ECI and EPI COPY files:

```
export COBCPY=/opt/cicscli/include
```

CICS for OS/2 Version 3 server implementation

For examples of programs that call the ECI and EPI, refer to the samples supplied with the CICS product. CICS for OS/2 V3.0 provides C, COBOL, and PL/I source and makefile samples in the following directories:

- \CICS300\TOOLS\C\SAMPLES\SOURCE
- \CICS300\TOOLS\COBOL\SAMPLES\SOURCE
- \CICS300\TOOLS\PLI\SAMPLES\SOURCE

CICS for OS/2 Version 3.1 provides C, COBOL, and PL/I source and makefile samples in the following directories:

- \CICS310\TOOLS\C\SAMPLES\SOURCE
- \CICS310\TOOLS\COBOL\SAMPLES\SOURCE
- \CICS310\TOOLS\PLI\SAMPLES\SOURCE

For 32-bit C programs:

```
icc /Gt+ prog.c faacic32.lib
```

For 16-bit C programs:

```
cl prog.c /Gs /link /stack:16384 faaclib
```

Creating applications

For COBOL programs:

If you are using IBM VisualAge for COBOL for OS/2:

```
cob2 -c -qapost -qnosequence -qlib. prog.cbl
```

```
ilink /NOFREE /NOD /NOI /MAP prog.obj,,OS2386 iwzrlib cclos232,prog.def
```

If you are using Micro Focus Object COBOL:

```
cobol prog.cbl;
```

```
cblnames -mPROG -oPROG.CBJ prog
```

```
link386 /NOD /NOI prog.obj prog.cbj,,OS2386 mfrts32 cclos232,prog.def
```

Note:

- You must resolve ECI and EPI function calls at link time using the system linkage convention. This is the default linkage for IBM VisualAge for COBOL for OS/2. If you use Micro Focus Object COBOL, you must use call-convention 8 on all ECI and EPI functions, or use the default call-convention 0 and compile using the LITLINK compiler directive.

For PL/I programs:

```
pli prog.pli ( tiled
```

```
link386 /NOD / NOI prog,,faacic32 ibmlink ceelink os2386
```

Chapter 5. External security interface

This chapter provides reference information about the external security interface (ESI).

The interface is described here in a language-independent manner, though the names chosen for the elements of the interface—functions, parameters, data structures, fields, constants, and so on—are similar to those provided for programming.

Any restrictions applicable to particular operating environments are identified under the functions to which they apply.

The chapter is organized as follows:

“Overview”

“Benefits of APPC PEM” on page 144

“Benefits of the ESI” on page 144

“ESI constants and data structures” on page 144

“ESI functions” on page 149

Overview

The External security interface (ESI) allows a non-CICS application to invoke services provided by advanced program-to-program communication (APPC) password expiration management (PEM).

APPC PEM with CICS provides support for an APPC architected sign-on transaction that signs on userids to a CICS server, and processes requests for a password change by:

- Identifying a user and authenticating that user’s identification
- Notifying specific users during the authentication that their passwords have expired
- Letting users change their passwords when (or before) the passwords expire
- Telling users how long their current passwords will remain valid
- Providing information about unauthorized attempts to access the server using a particular user identifier

Benefits of APPC PEM

APPC PEM has the following benefits:

- It enables users to update passwords on APPC links.
This can be particularly useful in the case of expired passwords. On APPC links that do not support APPC PEM, when users' passwords expire on remote systems, they are unable to update them from their own systems. The only alternative on a non-APPC PEM system is to log on directly to the remote system using a non-APPC link, such as an LU2 3270-emulation session, to update the password.
- It provides APPC users with additional information regarding their sign-on status; for example, the date and time at which they last signed on. It informs users whether their userid is revoked, or the password has expired, when they provide the correct password or PassTicket.

Benefits of the ESI

To use APPC PEM, you need a PEM client (requestor) and a PEM server linked by an APPC session. An external security manager (ESM), such as resource access control facility (RACF), or an equivalent ESM, must also be available to the PEM server.

The ESI provides the following functions:

- The **CICS_VerifyPassword** function, which allows a client application to verify that a password matches the password recorded by an ESM for a specified userid.
- The **CICS_ChangePassword** function, which allows a client application to change the password recorded by an ESM for a specified userid.
- The **CICS_SetDefaultSecurity** function, which allows a client application to specify a default userid and password to be used for ECI and EPI request to the server.

These functions allow a non-CICS application program to act as a PEM requestor without the application programmer having to manage an APPC conversation, which implies knowledge of the formats for PEM requests and replies, and of the interface to the local PEM server.

ESI constants and data structures

This section describes the constants and data structures that you will need to use the ESI. They are referred to in “ESI functions” on page 149.

ESI constants

The following constants are referred to symbolically in the descriptions of the ESI data structures, and functions in this chapter. Their values are given here to help you understand the descriptions. However, your code should always use the symbolic names of ESI constants provided for the programming language you are using.

Lengths of fields

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

ESI data structures

The following data structures are available for use with the ESI.

- **CICS_EsiDate_t**
- **CICS_EsiTime_t**
- **CICS_EsiDetails_t**

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

ESI constants and data structures

CICS_EsiDate_t

Purpose: The **CICS_EsiDate_t** structure contains a date represented as year, month, and day.

Fields:

Year 4-digit year held in **cics_ushort_t** format.

Month

Month held in **cics_ushort_t** format; values range from 1 to 12 with 1 representing January.

Day

Day held in **cics_ushort_t** format; values range from 1 to 31 with 1 representing the first day of the month.

CICS_EsiTime_t

Purpose: The **CICS_EsiTime** structure contains a time represented as hours, minutes, seconds, and hundredths of a second.

Fields:

Hours Hours held in **cics_ushort_t** format; values range from 0 to 23.

Minutes Minutes held in **cics_ushort_t** format; values range from 0 to 59.

Seconds Seconds held in **cics_ushort_t** format; values range from 0 to 59.

Hundredths Hundredths of a second held in **cics_ushort_t** format; values range from 0 to 99.

ESI constants and data structures

CICS_EsiDetails_t

Purpose: The **CICS_EsiDetails_t** structure contains information returned from a successful invocation of either the **CICS_VerifyPassword** or the **CICS_ChangePassword** functions.

Fields:

LastVerifiedDate

The date on which the password was last verified.

LastVerifiedTime

The time at which the password was last verified.

ExpiryDate

The date on which the password will expire.

ExpiryTime

The time at which the password will expire.

LastAccessDate

The date on which the userid was last accessed.

LastAccessTime

The time at which the userid was last accessed.

InvalidCount

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

ESI functions

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

- **CICS_VerifyPassword**
- **CICS_ChangePassword**
- **CICS_SetDefaultSecurity**

ESI functions

CICS_VerifyPassword

CICS_VerifyPassword	UserId Password System Details
---------------------	---

Purpose

The **CICS_VerifyPassword** function allows a client application to verify that a password matches the password recorded by an external security manager for a specified userid.

Note that the external security manager is assumed to be located in a server to which the client is connected.

Parameters

UserId

A pointer to a null-terminated string that specifies the userid whose password is to be verified. If the userid is shorter than CICS_ESI_USERID_MAX characters, it must be padded with nulls to a length of CICS_ESI_USERID_MAX+1.

The ESI uses this parameter only for input.

Password

A pointer to a null-terminated string that specifies the password to be checked by the external security manager for the specified userid. If the password is shorter than CICS_ESI_PASSWORD_MAX characters, it must be padded with nulls to a length of CICS_ESI_PASSWORD_MAX+1.

The ESI uses this parameter only for input.

System

A pointer to a null-terminated string that specifies the name of the server in which the password is to be verified. If the name is shorter than CICS_ESI_SYSTEM_MAX characters, it must be padded with nulls to a length of CICS_ESI_SYSTEM_MAX+1.

If the string is all nulls, then a server, currently the default server, is selected.

The ESI uses this parameter only for input.

Details

A pointer to the **CICS_EsiDetails_t** structure that on return contains further information returned by the external security manager.

The ESI uses the fields in this structure only for output.

Return codes**CICS_ESI_NO_ERROR**

The function completed successfully.

CICS_ESI_ERR_CALL_FROM_CALLBACK

The function was invoked from a callback routine.

CICS_ESI_ERR_SYSTEM_ERROR

An internal system error occurred.

CICS_ESI_ERR_NO_CICS

The client 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 client 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. You should consult the documentation for your client or server to see how to control the number of servers you can use.

CICS_ESI_ERR_MAX_SYSTEMS

You attempted to start requests to more servers than your configuration allows. You should consult the documentation for your client or server to see how to control the number of servers you can use.

CICS_ESI_ERR_NULL_USERID

The userid is set to nulls.

CICS_ESI_ERR_NULL_PASSWORD

The password is set to nulls.

ESI functions

CICS_ESI_ERR_PEM_NOT_SUPPORTED

Password expiry management is supported only for communications with the requested server over SNA and TCP62.

CICS_ESI_ERR_PEM_NOT_ACTIVE

The requested server does not support password expiry management.

CICS_ESI_ERR_PASSWORD_EXPIRED

The password has expired.

CICS_ESI_ERR_PASSWORD_INVALID

The password is invalid.

CICS_ESI_ERR_USERID_INVALID

The userid is not known to the external security manager.

CICS_ESI_ERR_SECURITY_ERROR

An error has been detected by the external security manager. The most likely explanation is that the userid has been revoked.

The mapping of actual return code values to the symbolic names is contained in the following files for CICS Universal Client for OS/2 and CICS Universal Client for Windows:

COBOL \copybook\cicsesi.cbl

C \include\cics_esi.h

PL/I \plihdr\cics_esi.inc

and in the following files for CICS Universal Client for AIX and CICS Universal Client for Solaris:

COBOL /include/CICSESI

C /include/cics_esi.h

CICS_ChangePassword

CICS_ChangePassword	UserId OldPassword NewPassword System Details
----------------------------	--

Purpose

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

Note that the external security manager is assumed to be located in a server to which the client is connected.

Parameters**UserId**

A pointer to a null-terminated string that specifies the userid whose password is to be changed. If the userid is shorter than **CICS_ESI_USERID_MAX** characters, it must be padded with nulls to a length of **CICS_ESI_USERID_MAX+1**.

The ESI uses this parameter only for input.

OldPassword

A pointer to a null-terminated string that specifies the current password for the specified userid. If the password is shorter than **CICS_ESI_PASSWORD_MAX** characters, it must be padded with nulls to a length of **CICS_ESI_PASSWORD_MAX+1**.

The ESI uses this parameter only for input.

NewPassword

A pointer to a null-terminated string that specifies the new password for the specified userid. If the password is shorter than **CICS_ESI_PASSWORD_MAX** characters, it must be padded with nulls to a length of **CICS_ESI_PASSWORD_MAX+1**.

The password is changed only if the currently password is correctly specified.

The ESI uses this parameter only for input.

ESI functions

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, then a server, currently the default server, is selected.

The ESI uses this parameter only for input.

Details

A pointer to the `CICS_EsiDetails_t` structure that on return contains further information returned by the external security manager.

The ESI uses the fields in this structure only for output.

Return codes

CICS_ESI_NO_ERROR

The function completed successfully.

CICS_ESI_ERR_CALL_FROM_CALLBACK

The function was invoked from a callback routine.

CICS_ESI_ERR_SYSTEM_ERROR

An internal system error occurred.

CICS_ESI_ERR_NO_CICS

The client 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 client 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. You should consult the documentation for your client or server to see how to control the number of servers you can use.

CICS_ESI_ERR_MAX_SYSTEMS

You attempted to start requests to more servers than your configuration allows. You should consult the documentation for your client or server to see how to control the number of servers you can use.

CICS_ESI_ERR_NULL_USERID

The userid is set to nulls.

CICS_ESI_ERR_NULL_OLDPASSWORD

The current password is set to nulls.

CICS_ESI_ERR_NULL_NEWPASSWORD

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 and TCP62.

CICS_ESI_ERR_PEM_NOT_ACTIVE

The requested server does not support password expiry management.

CICS_ESI_ERR_PASSWORD_INVALID

The password is invalid.

CICS_ESI_ERR_PASSWORD_REJECTED

The new password does not confirm to the standards defined for the external security manager.

CICS_ESI_ERR_USERID_INVALID

The userid is not known to the external security manager.

CICS_ESI_ERR_SECURITY_ERROR

An error has been detected by the external security manager. The most likely explanation is that the userid has been revoked.

The mapping of actual return code values to the symbolic names is contained in the following files for CICS Universal Client for OS/2 and CICS Universal Client for Windows:

COBOL \copybook\cicsesi.cbl

C \include\cics_esi.h

PL/I \plihdr\cics_esi.inc

and in the following files for CICS Universal Client for AIX and CICS Universal Client for Solaris:

COBOL /include/CICSESI

C /include/cics_esi.h

CICS_SetDefaultSecurity

CICS_SetDefaultSecurity	UserId
	Password
	System

Purpose

The **CICS_SetDefaultSecurity** function allows a client application to specify a default userid and password to be used for ECI and EPI requests passed to the server. The userid, and by implication the password, may be set to nulls.

The client application is responsible for verifying the userid and password.

Note that the userid and password, if required, may be obtained from any one of several places. The assumption is that the CICS Universal Client 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 NT userid, from the client's pop up window, and so on

Parameters

UserId

A pointer to a null-terminated string that specifies the userid to be set. If the userid is shorter than CICS_ESI_USERID_MAX characters, it must be padded with nulls to a length of CICS_ESI_USERID_MAX+1.

The ESI uses this parameter only for input.

Password

A pointer to a null-terminated string that specifies the password to be set for the specified userid. If the password is shorter than CICS_ESI_PASSWORD_MAX characters, it must be padded with nulls to a length of CICS_ESI_PASSWORD_MAX+1.

The ESI uses this parameter only for input.

System

A pointer to a null-terminated string that specifies the name of the server for which the password and userid are to be set. If

the name is shorter than CICS_ESI_SYSTEM_MAX characters, it must be padded with nulls to a length of CICS_ESI_SYSTEM_MAX+1.

If the string is all nulls, then a server, currently the default server, is selected.

The ESI uses this parameter only for input.

Return codes

CICS_ESI_NO_ERROR

The function completed successfully.

CICS_ESI_ERR_CALL_FROM_CALLBACK

The function was invoked from a callback routine.

CICS_ESI_ERR_SYSTEM_ERROR

An internal system error occurred.

CICS_ESI_ERR_NO_CICS

The client is unavailable, or the specified server is unavailable.

CICS_ESI_ERR_UNKNOWN_SERVER

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

CICS_ESI_ERR_USERID_INVALID

The length of the userid exceeds CICS_ESI_USERID_MAX.

CICS_ESI_ERR_PASSWORD_INVALID

The length of the password exceeds CICS_ESI_PASSWORD_MAX.

The mapping of actual return code values to the symbolic names is contained in the following files for CICS Universal Client for OS/2 and CICS Universal Client for Windows:

COBOL \copybook\cicsesi.cbl

C \include\cics_esi.h

PL/I \plihdr\cics_esi.inc

and in the following files for CICS Universal Client for AIX and CICS Universal Client for Solaris:

COBOL /include/CICSESI

C /include/cics_esi.h

ESI functions

Appendix A. ECI extensions that are environment-dependent

This chapter describes extensions to the ECI that are supported in certain environments. **They are not part of CICS Family Client/Server Programming.**

The chapter is organized as follows:

“Call type extensions”

“Time-outs” on page 161

“Fields to support ECI extensions” on page 162

“Reply message formats” on page 164

“ECI return notification” on page 164

“Summary of input parameter requirements” on page 165.

Call type extensions

The following call types are for asynchronous calls.

For more information about the program link calls, study Table 5 on page 164 in conjunction with “ECI_ASYNC call type” on page 27.

For more information about the status information calls, study Table 5 on page 164 in conjunction with “ECI_STATE_ASYNC call type” on page 39.

Asynchronous program link call, with notification by message (ECI_ASYNC_NOTIFY_MSG)

This call type is available only for programs running under OS/2 Presentation Manager, Microsoft Windows, or Windows NT.

The calling application gets control back when the ECI accepts the request. Note that this does not indicate that the program has started to run, merely that the parameters have been validated. The request might be queued for later processing.

The ECI sends a notification message to the specified window when the response is available. (For details of the message format, see “Reply message

Environment-dependent extensions

formats” on page 164.) On receipt of this notification, the calling application should use `ECI_GET_REPLY` or `ECI_GET_SPECIFIC_REPLY` to receive the actual response.

The following fields are required parameters for notification by message:

- Either `eci_async_notify.window_handle` (for OS/2 Presentation Manager or Windows NT environment) or `eci_async_notify.win_fields.hwnd` (for Microsoft Windows environment) identifies the window to be notified.
- `eci_async_notify.win_fields.hinstance` (for Microsoft Windows environment) indicates the `hInstance` of the calling program.
- `eci_message_id` indicates the message type to be used in the notification process.

`eci_message_qualifier` can be used as an input to provide a user-defined name for the call. It is returned as part of the notification message for the OS/2 Presentation Manager and Windows NT environments.

Asynchronous program link call, with notification by semaphore (`ECI_ASYNC_NOTIFY_SEM`)

This call type is available only for programs running under OS/2 or Windows NT.

The calling application gets control back when the ECI accepts the request. Note that this does not indicate that the program has started to run, merely that the parameters have been validated. The request might be queued for later processing.

The ECI posts the specified semaphore when the response is available. On receipt of this notification, the calling application should use `ECI_GET_REPLY` or `ECI_GET_SPECIFIC_REPLY` to receive the actual response.

`eci_message_qualifier` can be used as an input to provide a user-defined name for the call.

The following field is a required parameter for notification by semaphore:

- `eci_async_notify.sem_handle` refers to the semaphore.

Asynchronous status call, with notification by message (`ECI_STATE_ASYNC_MSG`)

This call type is available only for programs running under OS/2 Presentation Manager, Microsoft Windows, or Windows NT.

`eci_message_qualifier` can be used as an input to provide a user-defined name for the call.

The ECI sends a notification message to the specified window when the response is available. (For details of the message format, see “Reply message formats” on page 164.) On receipt of this notification, the calling application should use `ECI_GET_REPLY` or `ECI_GET_SPECIFIC_REPLY` to receive the actual response.

For details of the additional parameters relating to notification by message, see the description of the `ECI_ASYNC_NOTIFY_MSG` call type.

Asynchronous status call, with notification by semaphore (`ECI_STATE_ASYNC_SEM`)

This call type is available only for programs running under OS/2 or Windows NT.

`eci_message_qualifier` can be used as an input to provide a user-defined name for the call.

The ECI posts the specified semaphore when the response is available. On receipt of this notification, the calling application should use `ECI_GET_REPLY` or `ECI_GET_SPECIFIC_REPLY` to receive the actual response.

The following field is a required parameter for notification by semaphore:

- `eci_async_notify.sem_handle` refers to the semaphore.

Time-outs

The support consists of a field `eci_timeout` in the ECI parameter block, and two return codes: `ECI_ERR_RESPONSE_TIMEOUT` and `ECI_ERR_REQUEST_TIMEOUT`.

In the processing of timeouts, there are two cases to consider.

- The time-out occurs before the request has been forwarded to the server. In this case the response is `ECI_ERR_REQUEST_TIMEOUT`. The requested program was not called, and no server resources have been updated. If the call was intended to start, or be the whole of, a new logical unit of work, the logical unit of work was not started, and no recoverable resources have been updated. If the call was intended to continue an existing logical unit of work, the logical unit of work was continued, but no recoverable resources were updated, and the logical unit of work is still uncommitted. If the call was intended to end an existing logical unit of work, the logical unit of work was continued, no recoverable resources were updated, and the logical unit of work is still uncommitted.

Environment-dependent extensions

- The time-out occurs after the request has been forwarded to the server. In this case the response is `ECI_ERR_RESPONSE_TIMEOUT`, and it could be returned to a synchronous call, or to an asynchronous call, or to the reply solicitation request that gets the reply to an asynchronous call.
 - If the call was intended to be the only call of a new logical unit of work, the logical unit of work was started, but it is not possible for the application to determine whether updates were performed, and whether they were committed or backed out.
 - If the call was intended to end an existing logical unit of work by using `ECI_NO_EXTEND`, the logical unit of work has ended, but it is not possible for the application to determine whether updates were performed, and whether they were committed or backed out.
 - If the call was intended to continue or to end an existing logical unit of work by using `ECI_COMMIT`, the logical unit of work persists, and changes to recoverable resources are still pending.

Fields to support ECI extensions

The following fields in the ECI parameter block are to support environment-dependent extensions.

`eci_async_notify.window_handle`

(OS/2 and Windows NT environments, `ECI_ASYNC_NOTIFY_MSG` and `ECI_STATE_ASYNC_MSG` call types)

The handle of the window to which the reply message will be posted.

The ECI uses this field as input only.

Note: `eci_window_handle` is a synonym for this parameter.

`eci_async_notify.sem_handle`

(OS/2 and Windows NT environments, `ECI_ASYNC_NOTIFY_SEM` and `ECI_STATE_ASYNC_SEM` call types)

A reference to an OS/2 semaphore. This semaphore is cleared when the asynchronous response is ready for collection.

- 16-bit OS/2 applications should pass either a system semaphore handle or the address of a RAM semaphore.
- 32-bit OS/2 applications should pass an event semaphore handle.
- Windows NT applications should pass an event object handle.

The ECI uses this field as input only.

`eci_async_notify.win_fields.hwnd`

(Microsoft Windows environment, `ECI_ASYNC_NOTIFY_MSG` and `ECI_STATE_ASYNC_MSG` call types)

Environment-dependent extensions

The handle of the Microsoft Windows window to which the reply message will be posted.

The ECI uses this field as input only.

eci_async_notify.win_fields.hinstance

(Microsoft Windows environment, ECI_ASYNC_NOTIFY_MSG and ECI_STATE_ASYNC_MSG call types)

The Microsoft Windows hInstance of the calling program as supplied during program initialization.

The ECI uses this field as input only.

eci_sync_wait.hwnd

(Microsoft Windows environment, ECI_SYNC and ECI_STATE_SYNC call types)

The handle of the window that is to be disabled during the synchronous call.

The ECI uses this field as input only.

eci_message_id

(OS/2 Presentation Manager, Microsoft Windows, and Windows NT environments, ECI_ASYNC_NOTIFY_MSG and ECI_STATE_ASYNC_MSG call types)

The message identifier to be used for posting the reply message to the window specified in the relevant window handle.

The ECI uses this field as input only.

eci_timeout

(Program link calls and status calls only)

An integer field specifying the maximum time in seconds that a request from the application is allowed to take. A timeout occurs if the servicing of a request takes longer than the specified time. The value must be in the range 0 through 32767. A value of zero means that the application sets no limit on the time to service a request.

The ECI uses this field as input only.

eci_extend_mode

There is an additional value for program link calls in CICS OS/2 Version 1.20— ECI_CANCEL. It has the same effect as ECI_COMMIT.

Environment-dependent extensions

Reply message formats

When an application makes an asynchronous call requesting notification by message, the ECI returns the result in a message to a window using the specified window handle and message identifier.

For OS/2 Presentation Manager, the message is divided into two parameters, as follows:

MPARAM1

High-order 16 bits
Specified message qualifier

Low-order 16 bits
Return code

MPARAM2

4-character abend code, if applicable

For Microsoft Windows, the message is divided into two parameters, as follows:

wParam
Return code

lParam
4-character abend code, if applicable

For Windows NT, the message is divided into two parameters, as follows:

wParam
High-order 16 bits
Specified message qualifier

Low-order 16 bits
Return code

lParam
4-character abend code, if applicable

ECI return notification

Table 5. CICS_ExternalCall return codes — environment-dependent extensions

Return code	Meaning
ECI_ERR_NULL_WIN_HANDLE	An asynchronous call was specified with the window handle set to 0.

Table 5. *CICS_ExternalCall* return codes — environment-dependent extensions (continued)

Return code	Meaning
ECI_ERR_NULL_MESSAGE_ID	An asynchronous call was specified with the message identifier set to 0.
ECI_ERR_NULL_SEM_HANDLE	A null semaphore handle was passed when a valid handle was required.
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.

Summary of input parameter requirements

Table 6 on page 166 shows the input parameters for an ECI call, and, for each call type, whether the parameters are required (R), optional (O), or not applicable (-). Where a parameter is shown as optional or not-applicable an initial field setting of nulls is recommended. An asterisk (*) immediately following an R means that further details regarding applicability are given under the description of the parameter.

The following abbreviations are used in the **Parameter** column:

AN **async_notify**

WF **win_fields**

SW **sync_wait**

Also, all named parameters have an **eci_** prefix. Thus **AN.WF.hwnd** represents the **eci_async_notify.win_fields.hwnd** parameter.

The following 3-character abbreviations are used for the call types in the column headings of the table:

ANM **ECI_ASYNC_NOTIFY_MSG**

ANE **ECI_ASYNC_NOTIFY_SEM**

SAM **ECI_STATE_ASYNC_MSG**

SAE **ECI_STATE_ASYNC_SEM**

SYN **ECI_SYNC**

SSN **ECI_STATE_SYNC**

Environment-dependent extensions

Table 6. Input parameters for CICS_ExternalCall — environment-dependent extensions

Parameter, eci_	ANM	ANE	SAM	SAE	SYN	SSN
call_type	R	R	R	R	R	R
program_name	R*	R*	-	-	R*	-
userid	R	R	-	-	R	-
password	R	R	-	-	R	-
transid	O	O	-	-	O	-
commarea	O	O	R*	R*	O	R*
commarea_length	O	O	R*	R*	O	R*
timeout	O	O	O	O	O	O
extend_mode	R	R	R	R	R	R
AN.window_handle	R*	-	R*	-	-	-
AN.sem_handle	-	R	-	R	-	-
AN.WF.hwnd	R*	-	R*	-	-	-
AN.WF.hinstance	R*	-	R*	-	-	-
SW.hwnd	-	-	-	-	R*	R*
message_id	R	-	R	-	-	-
message_qualifier	O	O	O	O	O	O
luw_token	R	R	R*	R*	R	R*
version	O	O	O	O	O	O
system_name	O	O	O	O	O	O

Appendix B. CICS Universal Client Programming Samples

This chapter summarizes the samples provided for client/server programming in the CICS Universal Clients. The samples are provided in the following languages:

- C
- COBOL
- PL/I
- REXX

As far as possible the sample code adheres to the language standards, for example, ANSI C. The samples are all driven from the command line to avoid any dependence on platform-specific GUI code.

For each language, different levels of sample are provided. These include

- A simple sample to allow you to test that the Client is functioning and to give you a feel for the basic requirements of a Client application.
- More complex samples that demonstrate some of the more advanced and useful API features, and provides a more realistic example of a Client application.

Samples are provided for the ECI, EPI, and ESI.

For information on the languages and compilers supported on each client platform, refer to the appropriate *CICS Client Administration* book.

The samples are provided in the samples subdirectory of the product root directory. Under the samples directory there is a subdirectory for each programming language, for example, \samples\c.

The samples are described in detail, with compilation instructions and information on compiler considerations, in the SAMPLES.TXT file located in the samples directory.

Appendix C. ECI and EPI exits

This chapter describes exits you can add to the EPI and ECI when using IBM CICS Clients Version 2.0.3 and later. (CICS Client for DOS does support the ECI and EPI exits.) The exits allow you to influence the processing of ECI and EPI calls for certain application requests.

The chapter is organized as follows:

“Installing the exits”

“Exit routine environment” on page 170

“How the exit routines are described in the reference sections” on page 171

“ECI exits reference” on page 171

“Diagnostic information” on page 201

“CICSTERM, CICSPRNT and the EPI exits” on page 201

This chapter contains Product-sensitive Programming Interface and Associated Guidance Information.

Installing the exits

During ECI and EPI initialization, the client attempts to load the objects described in Table 7 from the CICS binary directory, and to call the corresponding entry points.

Table 7. ECI and EPI exits

	Object name	Entry point name
ECI	cicseciexit	CICS_EciExitInit
EPI	cicsepexit	CICS_EpiExitInit

Each entry point is passed a single parameter, a pointer to a structure that contains a list of addresses. The initialization code of the program puts the addresses of all the exits into the structure, and then the exits are called at appropriate points in ECI and EPI processing. Since the exits are entered by using the addresses supplied, you may give the exits any names you please, but in this manual conventional names are used for the exits.

If the objects are not found, no exit processing occurs.

The following files are supplied with your CICS Clients to help with programming the exits.

ECI and EPI exits

include\cicsecix.h

A header file that defines:

- inputs and outputs for each ECI exit
- the format of the list of addresses for calling ECI exits
- data structures used by ECI exits
- return code values for ECI exits.

samples\c\cicsecix.c

Skeleton code for **cicsecix**.

samples\c\cicsecix.def

Definition file for **cicsecix**.

samples\cicsecix.mak

A makefile to reconstruct **cicsecix**.

include\cicsepix.h

A header files that defines:

- inputs and outputs for each EPI exit
- the format of the list of addresses for calling EPI exits
- data structures used by EPI exits
- return code values for EPI exits.

samples\c\cicsepix.c

Skeleton code for **cicsepix**.

samples\c\cicsepix.def

Definition file for **cicsepix**.

samples\c\cicsepix.mak

A makefile to reconstruct **cicsepix**.

(On AIX and Solaris the pathnames follow the UNIX[®] conventions.)

Note: If you write a user-exit DLL, all exits must be included, apart from the following exceptions:

- **CICS_EciSetProgramAlias** is optional.
- Either **CICS_EpiTermIdExit** or **CICS_EpiTermIdInfoExit** should be included, however, any new DLLs should use **CICS_EpiTermIdInfoExit**.

Exit routine environment

In your exits:

- You must not make EPI or ECI calls.

- You should avoid waits or long-running code. (It is quite possible that the exit is running on the applications user interface thread, and thus any delays in returning could have a bad effect on the system’s responsiveness.)
- You must not register as an RPC server.
- You must follow the recommendations for multithreaded processes contained in Chapter 6 of *Guide to Writing DCE Applications*.

How the exit routines are described in the reference sections

The exit routines are described under the following headings:

- **Purpose**—describes the kind of processing that the exit is intended to perform.
- **When called**—describes where in ECI or EPI processing the exit is called.
- **Parameters**—describes the parameters supplied to the exit. Parameters are classified as follows:
 - Input—the exit may look at it, but must not change it.
 - Output—the exit must not look at it, but must store a value in it.
 - Input-output—the exit may look at it, and may store a value in it.
- **Return codes**—describes the possible values the exit can return to the ECI or EPI. In each case the subsequent behavior of the ECI or EPI is described.

ECI exits reference

In this section the following exits are discussed:

- **CICS_EciInitializeExit**
- **CICS_EciTerminateExit**
- **CICS_EciExternalCallExit1**
- **CICS_EciExternalCallExit2**
- **CICS_EciSystemIdExit**
- **CICS_EciDataSendExit**
- **CICS_EciDataReturnExit**
- **CICS_EciSetProgramAliasExit**

Table 8 summarizes the exit names, the parameters passed to each exit, and the possible return codes.

Table 8. Summary of ECI exits

Function name	Parameters	Return codes:
CICS_EciInitializeExit	Version Anchor	CICS_EXIT_OK CICS_EXIT_NO_EXIT CICS_EXIT_CANT_INIT_EXITS user-defined

ECI exits

Table 8. Summary of ECI exits (continued)

Function name	Parameters	Return codes:
CICS_EciTerminateExit	Anchor	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_STORAGE user-defined
CICS_EciExternalCallExit1	Anchor Token ParmPtr	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
CICS_EciExternalCallExit2	Anchor Token ParmPtr	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined
CICS_EciSystemIdExit	Anchor Token ParmPtr Reason	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM CICS_EXIT_GIVE_UP user_defined
CICS_EciDataSendExit	Anchor Token	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined
CICS_EciDataReturnExit	Anchor Token ParmPtr	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined
CICS_EciSetProgramAliasExit	Anchor EciParms Program	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user_defined

Identification token

In order for the exits to be able to relate calls for the same ECI request, an *identification token* is passed in as a parameter to all exits except **CICS_EciInitializeExit** and **CICS_EciTerminateExit**. The token is the same for **CICS_EciExternalCallExit1** and **CICS_EciExternalCallExit2** that relate to the same call, and on intervening **CICS_EciDataSendExit**, **CICS_EciDataReturnExit**, and **CICS_EciSystemIdExit** exits. (Note that **CICS_EciExternalCallExit1** and **CICS_EciExternalCallExit2** are not called for a reply solicitation request.)

The token is unique within the operating system that initiated the request, for the duration of the request. It may be reused once the last exit for the request has been called.

In the case of an extended logical unit of work, the token may be different on different requests within the logical unit of work. (Since we allow reuse of the token, and a new program link call may not be made until the ECI_GET_REPLY request for the previous asynchronous request has completed, it may also be the same.)

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

Process model implementation

All exits that relate to a particular request (i.e. have the same identification token) are called in the context of the application process.

CICS_EciInitializeExit

CICS_EciInitializeExit	Version Anchor
------------------------	-------------------

Purpose: To allow the user to set up an exit environment.

When called: On the first invocation of **CICS_ExternalCall**, for each process, after parameter validation has occurred.

Parameters:

Version Input parameter. The version of the ECI under which the exit is running.

Anchor Output parameter. A pointer to a pointer that will be passed to the ECI exits. The second pointer is not used by the ECI; it is passed to the exits as supplied. You can acquire storage in this exit and pass its address to the other exits.

Return codes:

CICS_EXIT_OK

The ECI continues processing this request, calling the exits where appropriate.

CICS_EXIT_NO_EXIT

The ECI continues processing this request, but does not call any more exits.

CICS_EXIT_CANT_INIT_EXITS

The ECI writes a CICS client 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 client trace record, and then continues processing this request, but does not call any more exits.

CICS_EciTerminateExit

CICS_EciTerminateExit	Anchor
-----------------------	--------

Purpose: To allow the user to clean up the exit environment. Any storage acquired by **CICS_EciInitializeExit** must be released in this exit.

CICS_EciTerminateExit is not called by the IBM CICS Clients.

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

The ECI writes a CICS client trace record, and then continues with termination.

CICS_EXIT_BAD_STORAGE

The ECI writes a CICS client 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 client trace record, and then continues with termination.

CICS_EciExternalCallExit1

CICS_EciExternalCallExit1	Anchor Token ParmPtr
---------------------------	----------------------------

Purpose: To allow the user to pick the best system to run the program. This exit is called exactly once on each program link and each status information call. It is not called on a reply solicitation call. Although the exit is called when **eci_luw_token** is not zero, any change it makes to **eci_system_name** is ignored, as the server was selected when the logical unit of work was started.

When called: On invocation of **CICS_ExternalCall**, for each program link call and each status information call, after the ECI has validated the parameters.

Parameters:

- Anchor** Input parameter. The pointer set up by **CICS_EciInitializeExit**.
- Token** Input parameter. The identification token established by the ECI for this request.
- ParmPtr** Input parameter. A pointer to the ECI parameter block. The exit must treat all fields in the ECI parameter block as inputs, except the **eci_system_name** field, which it may change.

Return codes:

CICS_EXIT_OK

The ECI continues to process the request with the **eci_system_name** now specified in the ECI parameter block.

CICS_EXIT_BAD_ANCHOR

The ECI writes a CICS client 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

The ECI writes a CICS client 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 client trace record, and then continues to process the request with the **eci_system_name** now specified in the ECI parameter block.

Notes: There is a limited set of conditions under which the exit may select a new system. The exit may select a system if the call is a program link or status information call, and if a new logical unit of work is being started. In other cases, the exit should return CICS_EXIT_OK.

If the calling application has put binary zeros as the system name in the parameter block, then the application is expecting that the system will be dynamically selected, and the exit may safely select the system.

If however the calling application has placed a system name in the parameter block, or if the application is a version 0 application, then it may not be expecting the target system to change, and application errors could result. In this case the exit would generally return without specifying a replacement system, with the result that the specified or default system name is to be used. If the exit chooses to change the selected system in this situation, then it may do so, but the following should be borne in mind.

- The exit routine must be sensitive to whether or not the modification of the target system will cause errors in the ECI application running on the client.
- The exit routine must maintain a knowledge base, keyed on appropriate data available to it, to enable it to determine whether this modification is acceptable to the client application.

CICS_EciExternalCallExit2

CICS_EciExternalCallExit2	Anchor Token ParmPtr
---------------------------	----------------------------

Purpose: To allow the user to see the results of synchronous ECI calls for information gathering purposes only. This exit is called exactly once on every application program link or status information call. It is not called on reply solicitation calls.

When called: Before the ECI call returns to the application, and after the return data is filled into the ECI parameter block.

Parameters:

- Anchor** Input parameter. The pointer set up by **CICS_EciInitializeExit**.
- Token** Input parameter. The identification token established by the ECI for this request.
- ParmPtr** Input parameter. A pointer to the ECI parameter block. The exit must treat all fields in the ECI parameter block as inputs.

Return codes:

CICS_EXIT_OK

The ECI returns control to the application that issued the **CICS_ExternalCall** request.

CICS_EXIT_BAD_ANCHOR

The ECI writes a CICS client trace record, and then returns control to the application that issued the **CICS_ExternalCall** request.

CICS_EXIT_BAD_PARM

The ECI writes a CICS client 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 client trace record, and then returns control to the application that issued the **CICS_ExternalCall** request.

CICS_EciSystemIdExit

CICS_EciSystemIdExit	Anchor Token ParmPtr Reason
----------------------	--

Purpose: To allow the user to supply a new system name when the name supplied in the ECI parameter block is not valid.

When called: This exit is called when an error occurs that may be corrected by selection of a new system, userid, or password. This is when the ECI would return a code of ECI_ERR_NO_CICS or ECI_ERR_UNKNOWN_SERVER or ECI_ERR_SECURITY_ERROR. It may be called when either when the client detects an error before data is sent to the server, or after data returns from the server.

Parameters:

Anchor	Input parameter. The pointer set up by CICS_EciInitializeExit .
Token	Input parameter. The identification token established by the ECI for this request.
ParmPtr	Input parameter. A pointer to the ECI parameter block. The exit must treat all fields in the ECI parameter block as inputs, except the following, which it may set: <ul style="list-style-type: none"> • eci_system_name • eci_userid • eci_password.
Reason	Input parameter. The reason code that explains why the application request has not so far succeeded.

Return codes:**CICS_EXIT_OK**

The ECI retries the application call using the new parameters in the ECI parameter block. (The CICS program communication area supplied by the application to the **CICS_ExternalCall** is preserved.) The application callback routine will not be called, nor will **CICS_EciExternalCallExit2**.

CICS_EXIT_BAD_ANCHOR

The ECI writes a CICS client trace record, and then returns to the application that issued the **CICS_ExternalCall** request.

ECI exits

CICS_EXIT_BAD_PARM

The ECI writes a CICS client 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 client trace record, and then retries the application call as described for **CICS_EXIT_OK**.

CICS_EciDataSendExit

CICS_EciDataSendExit	Anchor Token
-----------------------------	-------------------------

Purpose: To allow the user to time calls for performance analysis.

When called: As close as possible to the time that the request will be sent to the server.

Parameters:

Anchor Input parameter. The pointer set up by **CICS_EciInitializeExit**.

Token Input parameter. The identification token established by the ECI for this request.

Return codes:**CICS_EXIT_OK**

The ECI continues processing the request.

CICS_EXIT_BAD_ANCHOR

The ECI writes a client trace record, and then continues processing the request.

CICS_EXIT_BAD_PARM

The ECI writes a client 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 client trace record, and then continues processing the request.

CICS_EciDataReturnExit

CICS_EciDataReturnExit	Anchor Token ParmPtr
------------------------	----------------------------

Purpose: To allow the user to time calls for performance analysis.

When called: As close as possible to the time that the response from the server has been received, and the ECI block and commarea data for eventual return to the application has been built. It is also called if there is a timeout because of a lack of response from the server.

Parameters:

- Anchor** Input parameter. The pointer set up by **CICS_EciInitializeExit**.
- Token** Input parameter. The identification token established by the ECI for this request.
- ParmPtr** Input parameter. A pointer to the ECI parameter block. The exit must treat all fields in the ECI parameter block as inputs.

Return codes:

CICS_EXIT_OK

The ECI continues processing the request.

CICS_EXIT_BAD_ANCHOR

The ECI writes a client trace record, and then continues processing the request.

CICS_EXIT_BAD_PARM

The ECI writes a client 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 client trace record, and then continues processing the request.

CICS_EciSetProgramAliasExit

CICS_EciSetProgramAliasExit	Anchor EciParms Program
------------------------------------	--

Purpose: To allow the user to change the program name that the WorkLoad Manager of CICS Client for Windows NT uses for load balancing

This exit is only available when the WorkLoad Manager is enabled.

When called: Immediately before the WorkLoad Manager tries to select a server for an ECI program to connect to.

Parameters:

Anchor Input parameter. The pointer set up by **CICS_EciInitializeExit**.

ECIParms ECI parameter block.

Program The alias name of the ECI program that the WorkLoad Manager will use for load balancing.

Return codes:**CICS_EXIT_OK**

The ECI continues processing the request.

CICS_EXIT_BAD_ANCHOR

The ECI writes a client trace record, and then continues processing the request.

CICS_EXIT_BAD_PARM

The ECI writes a client 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 client trace record, and then continues processing the request.

EPI exits reference

In this section the following exits are discussed:

- **CICS_EpiInitializeExit**
- **CICS_EpiTerminateExit**
- **CICS_EpiAddTerminalExit**
- **CICS_EpiTermIdExit**
- **CICS_EpiTermIdInfoExit**
- **CICS_EpiStartTranExit**
- **CICS_EpiReplyExit**
- **CICS_EpiDelTerminalExit**
- **CICS_EpiGetEventExit**
- **CICS_EpiSystemIdExit**
- **CICS_EpiTranFailedExit**

Table 9 summarizes the exit names, the parameters passed to each exit, and the possible return codes.

Table 9. 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_EpiStartTranExit	Anchor TransId Data Size	CICS_EXIT_OK CICS_EXIT_BAD_ANCHOR CICS_EXIT_BAD_PARM user-defined

Table 9. 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

EPI exits

CICS_EpiInitializeExit

CICS_EpiInitializeExit	Version Anchor
------------------------	-------------------

Purpose

To allow the user to set up an exit environment.

When called

On each invocation of **CICS_EpiInitialize**, after the EPI has validated the parameters.

Parameters

- Version** Input parameter. The version of the EPI under which the exit is running.
- Anchor** Output parameter. A pointer to a pointer that will be passed to the EPI exits. The second pointer is not used by the EPI; it is passed to the exits as supplied. You can acquire storage in this exit and pass its address to the other exits.

Return codes

CICS_EXIT_OK

The EPI continues processing this request, calling the exits where appropriate.

CICS_EXIT_NO_EXIT

The EPI continues processing this request, but does not call any more exits.

CICS_EXIT_CANT_INIT_EXITS

The EPI writes a client 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 client trace record, and then continues processing this request, but does not call any more exits.

CICS_EpiTerminateExit

CICS_EpiTerminateExit	Anchor
-----------------------	--------

Purpose

To allow the user to clean up the exit environment. Any storage acquired by **CICS_EpiInitializeExit** must be released in this exit.

When called

On each invocation of **CICS_EpiTerminate**, after the EPI has validated the parameters.

Parameters

Anchor Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

Return codes**CICS_EXIT_OK**

Termination continues.

CICS_EXIT_BAD_ANCHOR

The EPI writes a client trace record, and then continues with termination.

CICS_EXIT_BAD_STORAGE

The EPI writes a client 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 client trace record, and then continues with termination.

CICS_EpiAddTerminalExit

CICS_EpiAddTerminalExit	Anchor NameSpace System NetName DevType
-------------------------	---

Purpose

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

When called

On each invocation of **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal**, after the EPI has validated the parameters.

Parameters

Anchor Input parameter. The pointer storage set up by **CICS_EpiInitializeExit**.

NameSpace Input-output parameter. On input, its value depends on the value supplied for the **NameSpace** parameter of the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call to which this exit relates:

- If a null pointer was supplied, this input is a pointer to a null string.
- If a non-null pointer was supplied, the **Namespace** input parameter points to a copy of this data.

On output, it will be used by the EPI in the same way as the value specified on the call would have been used.

System Input-output parameter. On input, it is the value supplied for the **System** parameter of the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call to which this exit relates. On output, it will be used by the EPI in the same way as the value specified on the call would have been used.

NetName Input-output parameter. On input, it is the value supplied for the **NetName** parameter of the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call to which this exit relates. On output, it will be used by the EPI in the same way as the value specified on the call would have been used.

DevType Input-output parameter. On input, it is the value supplied for the **DevType** parameter of the **CICS_EpiAddTerminal** or

CICS_EpiAddExTerminal call to which this exit relates. On output, it will be used by the EPI in the same way as the value specified on the call would have been used.

Return codes

CICS_EXIT_OK

Processing continues with the output values of **NameSpace**, **System**, **NetName**, and **DevType**.

CICS_EXIT_DONT_ADD_TERMINAL

The **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** is ended with a return code of **CICS_EPI_ERR_FAILED**. If the application uses **CICS_EpiGetSysError**, the value 109 is returned in the **Cause** field of the **CICS_EpiSysError_t** structure.

CICS_EXIT_BAD_ANCHOR

The EPI writes a client trace record, and then continues as for **CICS_EXIT_OK**.

CICS_EXIT_BAD_PARM

The EPI writes a client 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 client trace record, and then continues as for **CICS_EXIT_OK**.

Notes

Note on selection of systems:

If the calling application does not specify system name in its parameter list, then it is expecting that the system will be dynamically selected, and the exit may safely select the system.

If however the calling application specifies a system name, then it may not be expecting the target system to change and application errors could result. In this case the exit would generally not specify a replacement system, with the result that the specified or default system name, device type, etc. is to be used. If the exit chooses to change the selected system in this situation, then it may do so, but the following should be borne in mind.

- The exit routine must be sensitive to whether or not the modification of the target system will cause errors in the EPI application running on the client.
- The exit routine must maintain a knowledge base, keyed on appropriate data available to it, to enable it to determine whether this modification is acceptable to the client application.

EPI exits

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

CICS_EpiTermIdExit	Anchor TermIndex System
---------------------------	--

Purpose

To allow the user to know the terminal index allocated after a successful call to **CICS_EpiAddTerminal**.

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

When called

On each invocation of **CICS_EpiAddTerminal**, after the server has allocated the terminal.

Parameters

Anchor	Input parameter. The pointer set up by CICS_EpiInitializeExit .
TermIndex	Input parameter. This is the terminal index for the terminal resource just reserved or installed.
System	Input parameter. A pointer to a null-terminated string that specifies the name of the server in which the terminal resource has been reserved or installed.

Return codes**CICS_EXIT_OK**

Processing continues.

CICS_EXIT_BAD_ANCHOR

The EPI writes a client trace record, and then continues as for **CICS_EXIT_OK**.

CICS_EXIT_BAD_PARM

The EPI writes a client 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 client trace record, and then continues as for **CICS_EXIT_OK**.

CICS_EpiTermIdInfoExit

CICS_EpiTermIdInfoExit	Anchor Version TermIndex EpiDetails
------------------------	--

Purpose

To allow the user to retrieve information about the current terminal.

When called

Immediately after a CICS terminal has been installed

Parameters

- Anchor** Input parameter. The pointer set up by **CICS_EpiInitializeExit**.
- Version** Input parameter. The EPI version.
- TermIndex** Input parameter. The index of the terminal being installed.
- EpiDetails** Input parameter. A pointer to the **CICS_EpiDetails_t** structure, containing details about the terminal being installed.

Return codes

CICS_EXIT_OK

Processing continues.

CICS_EXIT_BAD_ANCHOR

The EPI writes a client trace record, and then continues as for **CICS_EXIT_OK**.

CICS_EXIT_BAD_PARM

The EPI writes a client 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 client trace record, and then continues as for **CICS_EXIT_OK**.

CICS_EpiStartTranExit

CICS_EpiStartTranExit	Anchor TransId Data Size
-----------------------	---

Purpose

To allow the user to see when a transaction is started, for information gathering purposes. This exit will not select a system, and has no return data.

When called

On invocation of **CICS_EpiStartTran**, after the EPI has validated the parameters.

Parameters

Anchor	Input parameter. The pointer set up by CICS_EpiInitializeExit .
TransId	Input parameter. The value supplied for the TransId parameter of the CICS_EpiStartTran call to which this exit relates.
Data	Input parameter. The value supplied for the Data parameter of the CICS_EpiStartTran call to which this exit relates.
Size	Input parameter. The value supplied for the Size parameter of the CICS_EpiStartTran call to which this exit relates.

Return codes**CICS_EXIT_OK**

Processing of the **CICS_EpiStartTran** call continues.

CICS_EXIT_BAD_ANCHOR

The EPI writes a client trace record, and then processing of the **CICS_EpiStartTran** call continues.

CICS_EXIT_BAD_PARM

The EPI writes a client 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 client trace record, and then processing of the **CICS_EpiStartTran** call continues.

EPI exits

CICS_EpiReplyExit

CICS_EpiReplyExit	Anchor TermIndex Data Size
-------------------	---

Purpose

To allow the user to see when a transaction is replied to, for information gathering purposes.

When called

On invocation of **CICS_EpiReply**, after the EPI has validated the parameters.

Parameters

- Anchor** Input parameter. The pointer set up by **CICS_EpiInitializeExit**.
- TermIndex** Input parameter. The value supplied for the **TermIndex** parameter of the **CICS_EpiReply** call to which this exit relates.
- Data** Input parameter. The value supplied for the **Data** parameter of the **CICS_EpiReply** call to which this exit relates.
- Size** Input parameter. The value supplied for the **Size** parameter of the **CICS_EpiReply** call to which this exit relates.

Return codes

CICS_EXIT_OK

Processing of the **CICS_EpiReply** call continues.

CICS_EXIT_BAD_ANCHOR

The EPI writes a client trace record, and then processing of the **CICS_EpiReply** call continues.

CICS_EXIT_BAD_PARM

The EPI writes a client 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 client trace record, and then processing of the **CICS_EpiReply** call continues.

CICS_EpiDelTerminalExit

CICS_EpiDelTerminalExit	Anchor TermIndex
-------------------------	---------------------

Purpose

To allow the user to clean up any terminal-related data structures.

When called

On invocation of **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal**, after the EPI has validated the parameters. To allow the user to clean up any terminal-related data structures.

Parameters

Anchor Input parameter. The pointer set up by **CICS_EpiInitializeExit**.

TermIndex Input parameter. The value supplied for the **TermIndex** parameter of the **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal** call to which this exit relates.

Return codes**CICS_EXIT_OK**

Processing of the **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal** call continues.

CICS_EXIT_BAD_ANCHOR

The EPI writes a client trace record, and then processing of the **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal** call continues.

CICS_EXIT_BAD_PARM

The EPI writes a client 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 client trace record, and then processing of the **CICS_EpiDelTerminal** or **CICS_EpiPurgeTerminal** call continues.

CICS_EpiGetEventExit

CICS_EpiGetEventExit	Anchor TermIndex Wait Event
----------------------	--

Purpose

To allow the user to collect data relating to the event that has arrived.

When called

Immediately before **CICS_EpiGetEvent** returns to the caller. The exit can then examine the data returned, time the response from the system, etc.

Parameters

- Anchor** Input parameter. The pointer set up by **CICS_EpiInitializeExit**.
- TermIndex** Input parameter. The value to be returned to the application in the **TermIndex** parameter of the **CICS_EpiGetEvent** call to which this exit relates.
- Wait** Input parameter. The value supplied for the **Wait** parameter of the **CICS_EpiGetEvent** call to which this exit relates.
- Event** Input parameter. The value to be returned to the application in the **Event** parameter of the **CICS_EpiGetEvent** call to which this exit relates.

Return codes

CICS_EXIT_OK

Processing of the **CICS_EpiGetEvent** call continues.

CICS_EXIT_BAD_ANCHOR

The EPI writes a client trace record, and then processing of the **CICS_EpiGetEvent** call continues.

CICS_EXIT_BAD_PARM

The EPI writes a client 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 client trace record, and then processing of the **CICS_EpiGetEvent** call continues.

CICS_EpiSystemIdExit

CICS_EpiSystemIdExit	Anchor NameSpace System NetName DevType FailedSystem Reason SubReason UserId PassWord
----------------------	--

Purpose

To allow the user to supply a new system name when the value supplied for **CICS_Epi_AddTerminal** or **CICS_EpiAddExTerminal** was invalid.

When called

Immediately before **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** returns to the application when an error occurred while trying to add the terminal. The error can be **CICS_EPI_ERR_SYSTEM**, **CICS_EPI_ERR_FAILED**, or **CICS_EPI_ERR_SERVER_DOWN**. It occurs whether or not **CICS_EpiAddTerminalExit** or **CICS_EpiAddExTerminal** has been called previously.

Note: On some systems the completion of **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** is returned to the application asynchronously, and in this case this exit will be called asynchronously.

Parameters

Anchor	Input parameter. The pointer set up by CICS_EpiInitializeExit .
NameSpace	Input-output parameter. The NameSpace parameter used in the failed CICS_EpiAddTerminal or CICS_EpiAddExTerminal .
System	Input-output parameter. The System parameter used in the failed CICS_EpiAddTerminal or CICS_EpiAddExTerminal .
NetName	Input-output parameter. The NetName parameter used in the failed CICS_EpiAddTerminal or CICS_EpiAddExTerminal .
DevType	Input-output parameter. The DevType parameter used in the failed CICS_EpiAddTerminal or CICS_EpiAddExTerminal .

EPI exits

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. If the reason is CICS_EPI_ERR_FAILED, this is the value that appears in the Cause field of the CICS_EpiSysError_t structure.
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 188 apply.

CICS_EXIT_DONT_ADD_TERMINAL

The **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** is ended with a return code of CICS_EPI_ERR_FAILED. If the application uses **CICS_EpiGetSysError**, the value 109 is returned in the **Cause** field of the **CICS_EpiSysError_t** structure.

CICS_EXIT_BAD_ANCHOR

The EPI writes a client trace record, and then the error that caused the exit to be called is returned to the application.

CICS_EXIT_BAD_PARM

The EPI writes a client 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 client trace record, and then the error that caused the exit to be called is returned to the application.

CICS_EpiTranFailedExit

CICS_EpiTranFailedExit	Anchor TermIndex Wait Event
------------------------	--------------------------------------

Purpose

To allow the user to collect data when a transaction abends or a terminal fails.

When called

Immediately before **CICS_EpiGetEvent** returns to the caller, with or without **GetEventExit**, when the event is CICS_EPI_EVENT_END_TRAN, and the **AbendCode** field is not blank.

Note that there are some failures on remote systems that can occur and will simply cause the presentation of a 3270 data stream with an error message and no abend code in the CICS_EPI_EVENT_END_TRAN. This error message may not even occur on the same event as the CICS_EPI_EVENT_END_TRAN. If the exit requires to handle this situation, it may monitor it through **CICS_EpiGetEventExit** and scan the appropriate 3270 data streams.

Parameters

Anchor	Input parameter. The pointer set up by CICS_EpiInitializeExit .
TermIndex	Input parameter. The value to be returned to the application in the TermIndex parameter of the CICS_EpiGetEvent call to which this exit relates.
Wait	Input parameter. The value supplied for the Wait parameter of the CICS_EpiGetEvent call to which this exit relates.
Event	Input parameter. The value to be returned to the application in the Event parameter of the CICS_EpiGetEvent call to which this exit relates.

Return codes**CICS_EXIT_OK**

Processing of the **CICS_EpiGetEvent** call continues.

CICS_EXIT_BAD_ANCHOR

The EPI writes a client trace record, and then processing of the **CICS_EpiGetEvent** call continues.

EPI exits

CICS_EXIT_BAD_PARM

The EPI writes a client 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 client trace record, and then processing of the **CICS_EpiGetEvent** call continues.

Diagnostic information

The CICS client traces the input parameters to the exits immediately before they are called, and the output of the exit when the exit returns. CICS tracing is not available for use within the exit.

CICSTERM, CICSPRNT and the EPI exits

With the CICS Universal Clients Version 3, the CICSTERM and CICSPRNT commands can be used to drive the EPI user exits. This can be used to provide similar load balancing functionality to the Load Manager of CICS Universal Clients for Windows 98 and Windows NT (see the *CICS Universal Client for Windows Administration* book).

To use the EPI exits, you supply a CICS_EPIEXITINIT function in a DLL called *cicsepix.dll* (*cicsepix.a* on UNIX platforms.) When a CICSTERM or CICSPRNT session is started, the CICS Universal Client looks for a *cicsepix.dll* (*cicsepix.a*) in its bin directory. If no DLL is found, no exit processing occurs.

The CICS_EPIEXITINIT function sets an ExitList structure to point to the addresses of all the exit functions, which are also contained in *cicsepix.dll*. The sample CICS_EPIEXITINIT is as follows:

```
void CICSEXIT CICS_EPIEXITINIT(CICS_EpiExitList_t *ExitList)
{
    ExitList->InitializeExit    = &CICS_EpiInitializeExit;
    ExitList->TerminateExit     = &CICS_EpiTerminateExit;
    ExitList->AddTerminalExit   = &CICS_EpiAddTerminalExit;
    ExitList->StartTranExit     = &CICS_EpiStartTranExit;
    ExitList->ReplyExit         = &CICS_EpiReplyExit;
    ExitList->DelTerminalExit   = &CICS_EpiDelTerminalExit;
    ExitList->GetEventExit      = &CICS_EpiGetEventExit;
    ExitList->TranFailedExit    = &CICS_EpiTranFailedExit;
    ExitList->SystemIdExit      = &CICS_EpiSystemIdExit;
    ExitList->TermIdExit        = &CICS_EpiTermIdExit;
    ExitList->TermIdInfoExit    = &CICS_EpiTermIdInfoExit;
}
```

As the exits are entered by using the addresses supplied, you can give them any name you want, as long as their function signature is exactly the same as the CICS_Epi* functions. Therefore, the ExitList-> exits could be considered as generic terminal control exits.

Initializeexit is passed a version number of X'FF000000' when driven by CICSTERM or CICSPRNT. This enables user programs to be able to differentiate between CICSTERM and CICSPRNT user exits, and EPI user exits if they wish to do so.

Diagnostic information

CICSTERM and CICSPRNT also drive the EPI tracepoints.

The following describes the subset of the EPI exits that are available for CICSTERM and CICSPRNT and how they are implemented. How they affect the CICS Clients terminal emulator behavior is described.

EPI: CICS_EpiInitializeExit

This EPI exit does not affect the running of the calling EPI program, but it does allow the user to switch the user exits on or off for the process that calls it. It is called once per process that uses the EPI. It is called before any other EPI calls take place, and is called at the end of a successful **CICS_EpiInitialize**.

CICSTERM: InitializeExit

It is important that this exit is called once only for each CICSTERM session that is created, because each CICSTERM runs in a separate process. The version number passed is 'X'FF000000'.

EPI: CICS_EpiTerminateExit

Called by **CICS_EpiTerminate**, this is always the last EPI call in a particular process. It does not affect the running of the calling EPI program. It is called after checking that the EPI was initialized, and that there is not an active notify thread, but just before EPI is actually terminated. The EPI exit DLL is unloaded immediately following the user exit call.

CICSTERM: TerminateExit

Only called once during CICSTERM termination.

EPI: CICS_EpiAddTerminalExit

Allows the user to select a server, change the server parameters passed to the EPI call, and refuse to add a terminal to a server. This all happens from within the EPI call. The EPI program subsequently refers to the server by an index number, therefore the program does not need to know what server it is actually connected to. If the user exit refuses to connect a server, then **CICS_EpiSystemIDExit** is not called (see below for further details). **CICS_EpiAddTerminalExit** is called after **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** has verified that the EPI has been successfully initialized, and that there is a free session. It is called before the **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call actually sends the terminal definition to the server.

CICSTERM: AddTerminalExit

The /s or /r parameters of CICSTERM allow the user to specify that the CICS Universal Client can connect to:

- The first server defined in the client configuration file.
- A server chosen by the user from a list of available servers
- A server specified by the /s or /r parameter.

AddTerminalExit receives the system name as a parameter, and can specify a different server if required, or reject the server and cause the terminal emulator to terminate. If **AddTerminalExit** rejects the install, CICSTERM displays an error to the effect that the server is unavailable.

EPI: CICS_EpiSystemIdExit

Allows the user to re-select a server if a **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** fails. This user exit is called if a **CICS_EpiAddTerminal** fails (but not if the **CICS_EpiAddTerminalExit** causes the failure). If it returns **CICS_EXIT_OK**, **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** tries to add the terminal again. The server parameters can be changed by this exit between retries.

CICS_EpiSystemIdExit can be called asynchronously or synchronously by EPI programs. **CICS_EpiSystemIdExit** can be presented with:

- A **CICS_EPI_ERR_SYSTEM** error, meaning the server is unknown, or,
- A **CICS_EPI_ERR_SERVER_DOWN** error, meaning the server has failed, or,
- A **CICS_EPI_ERR_SECURITY** error, for a security failure, or,
- A **CICS_EPI_ERR_FAILED** error for any other type of failure.

It is also passed a parameter that is the same as the **cics_syserr_t** data structure **cause** field. This value further specifies the error and is a value specific to the operating environment

CICSTERM: SystemIdExit

If a CICSTERM **CICS_EpiAddTerminal** or **CICS_EpiAddExTerminal** call fails due to the client requester not having enough sessions free (governed by the **Maximum Requests** parameter in the client configuration file) then **SystemIdExit** is called with **CICS_EPI_ERR_FAILED** as the primary reason code and 7046 as the secondary reason code. The secondary reason code number is the same as the client trace number for a resource shortage (CCL7046E). In all other cases of **CICS_EPI_ERR_FAILED**, CICSTERM passes a secondary reason code of 0.

If no user exits are active, then CICSTERM retries a terminal install if it fails due to there not being enough available sessions. (This allows terminals to wait for free sessions before being installed.) If there are user exits available, then any retry behavior is controlled completely by the exit.

Diagnostic information

CICS_EpiSystemId is always called synchronously from the terminal thread, that is, the terminal install response is sent, it waits for a reply, and then drives **SystemIdExit** if the response is bad.

EPI: CICS_EpiTermIdExit

Allows the user to know what EPI Termid an added terminal is given. This is only called after a terminal has been successfully installed on a server. It does not affect the running of the EPI program. EPI Termid numbers are local to each process the EPI program runs under.

CICSTERM: TermIdExit

As only one CICSTERM runs per process, the Termid number is hardcoded to 1.

EPI: CICS_EpiTermIdInfoExit

Allows the user to know details about a terminal. This is called after a terminal has been successfully installed on a server.

CICSTERM: TermIdInfoExit

As only one CICSTERM runs per process, the Termid number is hardcoded to 1.

EPI: CICS_EpiDelTerminalExit

Allows the user to clean up terminal code. It is called when **CICS_EpiDelTerminal** is issued. It does not affect the running of the EPI program.

CICSTERM: DelTerminalExit

As only one CICSTERM runs per process, the Termid number is hardcoded to 1. It is called just before **TerminateExit** when the terminal is ended. When the server fails the **AddTerminalExit** is called again when it is restarted. However the **DelTerminalExit** is not called when the server fails.

EPI: CICS_EpiStartTranExit

Allows a user to see that a transaction has been started, and to see the Transid & 3270 data sent to it. It does not affect the running of the EPI program. **CICS_EpiStartTranExit**; is called after the EPI state has been verified, and just before the **CICS_EpiStartTran** is called.

Note that a pseudo-conversational transaction causes the **CICS_EpiStartTranExit**; to be called because in this case an EPI program would have to start a transaction again.

CICSTERM: StartTranExit:

If a non-ATI transaction is being started, then **StartTranExit** is called, sending a blank in the Transid field and the TIOA (terminal input output area) for the Data field. The Transid is either the first four characters of the TIOA data, or it will follow a 3270 Set Buffer

Address (SBA) command (which begins X'11'). In the latter case, it will start on the 4th byte of the TIOA (as a SBA command takes up a total of three bytes).

StartTranExit is not driven for ATI transactions. This is because the exit is normally driven by the **CICS_EpiStartTran** API call, and this call is not made to start ATI transactions. However pseudo-conversational transactions will drive **StartTranExit**. This is because in an EPI program, the user would have to call **CICS_EpiStartTran** to start the pseudo-conversational transaction. In the case of pseudo-conversational transactions, the transaction id is put in the transid parameter block and the TIOA passed in the data block does not contain the transaction id.

StartTranExit is not called as a result of an EXEC CICS RETURN TRANSID*name* IMMEDIATE command issued by an application from a CICSTERM session.

EPI: CICS_EpiReplyExit

Allows the user to see when an application is replied to. It does not affect the running of the EPI program.

CICSTERM ReplyExit

Activated when the client is sending data and a transaction is currently active. The Termid number is hard coded to 1. The terminal TIOA is passed to **ReplyExit**

ReplyExit is not called as a result of an EXEC CICS RETURN TRANSID*name* IMMEDIATE command issued by an application from a CICSTERM session.

The **GetEventExit** and **TranFailedExit** exits are not implemented for CICSTERM and CICSPRNT

Appendix D. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply in the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to 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. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Programming interface information

This book is intended to help you to write application programs that use the features provided by various members of the CICS family. This book documents General-use Programming Interface and Associated Guidance Information provided by various members of the CICS family.

General-use programming interfaces allow the customer to write programs that obtain the services of various members of the CICS family.

This book also documents Product-sensitive Programming Interface and Associated Guidance Information provided by various members of the CICS family.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of CICS. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs, by an introductory statement to a chapter or section.

Trademarks

The following terms, used in this publication, are trademarks or service marks of IBM Corporation in the United States, or other countries, or both:

AIX	CICS	CICS/ESA
CICS/MVS	CICS OS/2	CICS/VSE
CICS/400	IBM	OS/2
OS/390	OS/400	Presentation Manager
VisualAge	System/390	

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

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Index

A

asynchronous calls 3
ATISState parameter
 CICS_EpiATISState function 106
Attributes parameter
 CICS_EpiAddExTerminal
 function 93
autoinstall 59

B

BMS paging 57

C

callback routine
 ECI 27, 32, 39, 42
 EPI 63, 64, 88, 93
CICS_ChangePassword function
 definition 153
CICS_ECI_DESCRIPTION_MAX 55
CICS_ECI_SYSTEM_MAX 55
CICS_EciDataReturnExit 182
CICS_EciDataSendExit 181
CICS_EciExitInit entry point 169
CICS_EciExternalCallExit1 176
CICS_EciExternalCallExit2 178
CICS_EciInitializeExit 174
CICS_EciListSystems function 8, 55
 ECI_ERR_INVALID_DATA
 _LENGTH 56
 ECI_ERR_MORE_SYSTEMS 56
 ECI_ERR_NO_CICS 56
 ECI_ERR_NO_SYSTEMS 56
 ECI_ERR_SYSTEM_ERROR 56
 ECI_NO_ERROR 56
CICS_EciSetProgramAliasExit 183
CICS_EciSystem_t data structure
 definition 55
 use 55
CICS_EciSystemIdExit 179
CICS_EciTerminateExit 175
CICS_EPI_ADD_TERM event
 definition 115
CICS_EPI_ATI_HOLD 106
CICS_EPI_ATI_ON 106
CICS_EPI_ATI_QUERY 106
CICS_EPI_DESCRIPTION_MAX, 69
CICS_EPI_DEVTYPE_MAX 87, 92
CICS_EPI_END_FAILED 120
CICS_EPI_END_OUTSERVICE 120
CICS_EPI_END_SHUTDOWN 120

CICS_EPI_END_SIGNOFF 120
CICS_EPI_END_UNKNOWN 120
CICS_EPI_ERR_ABENDED return
 code
 CICS_EpiReply function 105
CICS_EPI_ERR_ADDTYPE_INVALID
 return code
 CICS_EpiAddExTerminal
 function 94
CICS_EPI_ERR_ALREADY_INSTALLED
 return code
 CICS_EpiAddExTerminal
 function 94
 CICS_EpiAddTerminal
 function 89
CICS_EPI_ERR_ATI_ACTIVE return
 code
 CICS_EpiStartTran function 102
CICS_EPI_ERR_ATI_STATE return
 code
 CICS_EpiATISState function 106
CICS_EPI_ERR_BAD_INDEX return
 code
 CICS_EpiATISState function 107
 CICS_EpiDelTerminal
 function 97
 CICS_EpiGetEvent function 111
 CICS_EpiGetSysError
 function 112
 CICS_EpiInquireSystem
 function 96
 CICS_EpiPurgeTerminal
 function 98
 CICS_EpiReply function 104
 CICS_EpiSenseCode
 function 108
CICS_EPI_ERR_CC SID_INVALID
 return code
 CICS_EpiAddExTerminal
 function 95
 CICS_EpiAddTerminal
 function 89
CICS_EPI_ERR_FAILED return code
 CICS_EpiAddExTerminal
 function 93
 CICS_EpiAddTerminal
 function 88

CICS_EPI_ERR_FAILED return
 code (*continued*)
 CICS_EpiATISState function 107
 CICS_EpiDelTerminal
 function 97
 CICS_EpiGetEvent function 111
 CICS_EpiGetSysError
 function 112, 113
 CICS_EpiInitialize function 82
 CICS_EpiInquireSystem
 function 96
 CICS_EpiListSystems
 function 84
 CICS_EpiPurgeTerminal
 function 98
 CICS_EpiReply function 104
 CICS_EpiSenseCode
 function 108
 CICS_EpiStartTran function 102
 CICS_EpiTerminate function 83
CICS_EPI_ERR_IN_CALLBACK
 return code
 CICS_EpiAddExTerminal
 function 94
 CICS_EpiAddTerminal
 function 89
 CICS_EpiATISState function 107
 CICS_EpiDelTerminal
 function 97
 CICS_EpiGetEvent function 111
 CICS_EpiInquireSystem
 function 96
 CICS_EpiListSystems
 function 85
 CICS_EpiPurgeTerminal
 function 98
 CICS_EpiReply function 105
 CICS_EpiSenseCode
 function 108
 CICS_EpiSetSecurity
 function 100
 CICS_EpiStartTran function 103
 CICS_EpiTerminate function 83
CICS_EPI_ERR_IS_INIT return code
 CICS_EpiInitialize function 82
CICS_EPI_ERR_MAX_SESSIONS
 return code
 CICS_EpiAddExTerminal
 function 95

CICS_EPI_ERR_MAX_SESSIONS return code (<i>continued</i>) CICS_EpiAddTerminal function 90 CICS_EpiStartTran function 103	CICS_EPI_ERR_NOT_INIT return code 67 (<i>continued</i>) CICS_EpiAddTerminal function 89 CICS_EpiATISState function 107 CICS_EpiDelTerminal function 97 CICS_EpiGetEvent function 111 CICS_EpiGetSysError function 112 CICS_EpiInquireSystem function 96 CICS_EpiListSystems function 85 CICS_EpiPurgeTerminal function 98 CICS_EpiReply function 105 CICS_EpiSenseCode function 108 CICS_EpiSetSecurity function 100 CICS_EpiStartTran function 103 CICS_EpiTerminate function 83	CICS_EPI_ERR_RESOURCE _SHORTAGE return code (<i>continued</i>) CICS_EpiAddTerminal function 89 CICS_EpiStartTran function 103 CICS_EPI_ERR_RESPONSE_TIMEOUT return code CICS_EpiAddExTerminal function 94 CICS_EPI_ERR_SECURITY return code CICS_EpiAddExTerminal function 93 CICS_EpiAddTerminal function 89 CICS_EPI_ERR_SENSE_CODE return code CICS_EpiSenseCode function 108 CICS_EPI_ERR_SERVER_BUSY return code CICS_EpiAddExTerminal function 95 CICS_EpiAddTerminal function 89 CICS_EPI_ERR_SERVER_DOWN return code CICS_EpiAddExTerminal function 94 CICS_EpiAddTerminal function 89 CICS_EpiReply function 105 CICS_EpiStartTran function 103 CICS_EPI_ERR_SIGNON_NOT_POSS return code CICS_EpiAddExTerminal function 94 CICS_EPI_ERR_SIGNONCAP_INVALID return code CICS_EpiAddExTerminal function 94 CICS_EPI_ERR_SYSTEM_ERROR return code CICS_EpiSetSecurity function 100 CICS_EPI_ERR_SYSTEM return code CICS_EpiAddExTerminal function 93 CICS_EpiAddTerminal function 89 CICS_EPI_ERR_TERMID_INVALID return code CICS_EpiAddExTerminal function 94
CICS_EPI_ERR_MAX_SYSTEMS return code CICS_EpiAddExTerminal function 95 CICS_EpiAddTerminal function 90	CICS_EPI_ERR_NULL_PARM return code CICS_EpiAddExTerminal function 94 CICS_EpiAddTerminal function 89	
CICS_EPI_ERR_MAX_TERMS return code CICS_EpiAddTerminal function 88	CICS_EPI_ERR_NULL_PASSWORD return code CICS_EpiSetSecurity function 100	
CICS_EPI_ERR_MODEL_INVALID return code CICS_EpiAddExTerminal function 94 CICS_EpiAddTerminal function 89	CICS_EPI_ERR_NULL_USERID return code CICS_EpiSetSecurity function 100	
CICS_EPI_ERR_MORE_DATA return code CICS_EpiGetEvent function 111	CICS_EPI_ERR_PASSWORD_INVALID return code CICS_EpiAddExTerminal function 94 CICS_EpiSetSecurity function 100	
CICS_EPI_ERR_MORE_EVENTS return code CICS_EpiGetEvent function 111	CICS_EPI_ERR_RESOURCE _SHORTAGE return code CICS_EpiAddExTerminal function 94 CICS_EpiSetSecurity function 100	
CICS_EPI_ERR_MORE_SYSTEMS return code CICS_EpiListSystems function 84	CICS_EPI_ERR_SYSTEM_ERROR return code CICS_EpiSetSecurity function 100	
CICS_EPI_ERR_NO_CONVERSE return code CICS_EpiReply function 104	CICS_EPI_ERR_SYSTEM return code CICS_EpiAddExTerminal function 93 CICS_EpiAddTerminal function 89	
CICS_EPI_ERR_NO_DATA return code CICS_EpiReply function 104 CICS_EpiStartTran function 103	CICS_EPI_ERR_TERMID_INVALID return code CICS_EpiAddExTerminal function 94	
CICS_EPI_ERR_NO_EVENT return code CICS_EpiGetEvent function 111		
CICS_EPI_ERR_NO_SYSTEMS return code CICS_EpiListSystems function 85		
CICS_EPI_ERR_NOT_3270_DEVICE return code CICS_EpiAddExTerminal function 94 CICS_EpiAddTerminal function 89		
CICS_EPI_ERR_NOT_INIT return code 67 CICS_EpiAddExTerminal function 93		

CICS_EPI_ERR_TERMID_INVALID return code (*continued*)
 CICS_EpiAddTerminal function 89

CICS_EPI_ERR_TRAN_ACTIVE return code
 CICS_EpiDelTerminal function 97

CICS_EPI_ERR_TTI_ACTIVE return code
 CICS_EpiStartTran function 103

CICS_EPI_ERR_USERID_INVALID return code
 CICS_EpiAddExTerminal function 94
 CICS_EpiSetSecurity function 100

CICS_EPI_ERR_VERSION return code
 CICS_EpiAddExTerminal function 95
 CICS_EpiInitialize function 82
 CICS_EpiPurgeTerminal function 98
 CICS_EpiSetSecurity function 100

CICS_EPI_ERR_WAIT return code
 CICS_EpiGetEvent function 111

CICS_EPI_ERROR_MAX 76

CICS_EPI_EVENT_ADD_TERM event
 use 64

CICS_EPI_EVENT_CONVERSE event
 definition 117
 use 64, 65, 104, 117, 123

CICS_EPI_EVENT_END_TERM event
 definition 120
 use 64, 74, 97, 98

CICS_EPI_EVENT_END_TRAN event
 definition 118
 use 64, 74, 101

CICS_EPI_EVENT_SEND event
 definition 116
 use 64, 65, 123

CICS_EPI_EVENT_START_ATI event
 definition 119
 use 64, 65, 102

CICS_EPI_NETNAME_MAX 72, 87, 91

CICS_EPI_NORMAL return code
 CICS_EpiAddExTerminal function 95

CICS_EPI_NORMAL return code (*continued*)
 CICS_EpiAddTerminal function 90
 CICS_EpiATISState function 107
 CICS_EpiDelTerminal function 97
 CICS_EpiGetEvent function 111
 CICS_EpiGetSysError function 113
 CICS_EpiInitialize function 82
 CICS_EpiInquireSystem function 96
 CICS_EpiListSystems function 85
 CICS_EpiPurgeTerminal function 98
 CICS_EpiReply function 105
 CICS_EpiSenseCode function 109
 CICS_EpiSetSecurity function 100
 CICS_EpiStartTran function 103
 CICS_EpiTerminate function 83

CICS_EPI_NOWAIT 63, 110

CICS_EPI_NULL_PARAM return code
 CICS_EpiATISState function 107

CICS_EPI_READTIMEOUT_EXPIRED 118

CICS_EPI_SENSE_OPCHECK 108

CICS_EPI_SENSE_REJECT 108

CICS_EPI_SYSTEM_MAX 69, 86, 91

CICS_EPI_TERM_INDEX_NONE 110

CICS_EPI_TRAN_NO_ERROR 118

CICS_EPI_TRAN_NOT_STARTED 118

CICS_EPI_TRAN_STATE_UNKNOWN 118

CICS_EPI_TRANSID_MAX 74, 101

CICS_EPI_VERSION_200 82

CICS_EPI_VERSION return code
 CICS_EpiGetSysError function 113
 CICS_EpiSenseCode function 109

CICS_EPI_WAIT 63, 110

CICS_EpiAddExTerminal function
 definition 91
 use 59, 63, 69, 70, 72

CICS_EpiAddTerminal function
 definition 86
 use 59, 63, 69, 72, 101, 106, 112, 125, 126

CICS_EpiAddTerminalExit 188

CICS_EpiATISState function
 definition 106

CICS_EpiAttributes_t data structure
 definition 70

CICS_EpiDelTerminal function
 definition 97
 use 61, 83, 97, 120

CICS_EpiDelTerminalExit 195

CICS_EpiDetails_t data structure
 definition 72
 use 86, 88, 91, 93

CICS_EpiEventData_t data structure
 definition 74
 use 63, 64, 110, 111

CICS_EpiExitInit entry point 169

CICS_EpiGetEvent function
 definition 110
 use 63, 64, 74, 75, 97, 102, 125

CICS_EpiGetEventExit 196

CICS_EpiGetSysError function
 definition 112
 use 76, 126

CICS_EpiInitialize function
 definition 82
 for Microsoft Windows environment 125
 use 58, 59, 83, 84, 112, 126

CICS_EpiInitializeExit 186

CICS_EpiInquireSystem function
 definition 96

CICS_EpiListSystems function
 definition 84
 use 59, 69, 112, 132

CICS_EpiPurgeTerminal function
 definition 98
 use 61

CICS_EpiReply function
 definition 104
 use 65, 117, 122

CICS_EpiReplyExit 194

CICS_EpiSenseCode function
 definition 108

CICS_EpiSetSecurity function
 definition 99

CICS_EpiStartTran function
 definition 101
 use 62, 64, 65, 66, 102, 118, 122

CICS_EpiStartTranExit 193

CICS_EpiSysError_t data structure
 definition 76
 use 112

CICS_EpiSystem_t data structure
 definition 69
 use 84, 132

CICS_EpiSystemIdExit 197

CICS_EpiTermIdExit 191	CICS_ESI_ERR_NULL_PASSWORD	CICS_ESI_ERR_SYSTEM_ERROR
CICS_EpiTermIdInfoExit 192	return code	return code (<i>continued</i>)
CICS_EpiTerminate function	CICS_VerifyPassword	CICS_SetDefaultSecurity
definition 83	function 151	function 157
use 58, 112	CICS_ESI_ERR_NULL_USERID	CICS_VerifyPassword
CICS_EpiTerminateExit 187	return code	function 151
CICS_EpiTranFailedExit 199	CICS_ChangePassword	CICS_ESI_ERR_UNKNOWN_SERVER
CICS_ESI_ERR_CALL_	function 155	return code
FROM_CALLBACK return code	CICS_VerifyPassword	CICS_ChangePassword
CICS_ChangePassword	function 151	function 154
function 154	CICS_ESI_ERR_PASSWORD	CICS_SetDefaultSecurity
CICS_SetDefaultSecurity	_REJECTED return code	function 157
function 157	CICS_ChangePassword	CICS_VerifyPassword
CICS_VerifyPassword	function 155	function 151
function 151	CICS_ESI_ERR_PASSWORD_EXPIRED	CICS_ESI_ERR_USERID_INVALID
CICS_ESI_ERR_CICS_DIED return	return code	return code
code	CICS_VerifyPassword	CICS_ChangePassword
CICS_ChangePassword	function 152	function 155
function 154	CICS_ESI_ERR_PASSWORD_INVALID	CICS_SetDefaultSecurity
CICS_VerifyPassword	return code	function 157
function 151	CICS_ChangePassword	CICS_VerifyPassword
CICS_ESI_ERR_MAX_SESSIONS	function 155	function 152
return code	CICS_SetDefaultSecurity	CICS_ESI_NO_ERROR return code
CICS_ChangePassword	function 157	CICS_ChangePassword
function 154	CICS_VerifyPassword	function 154
CICS_VerifyPassword	function 152	CICS_SetDefaultSecurity
function 151	CICS_ESI_ERR_PEM_NOT	function 157
CICS_ESI_ERR_MAX_SYSTEMS	_SUPPORTED return code	CICS_VerifyPassword
return code	CICS_ChangePassword	function 151
CICS_ChangePassword	function 155	CICS_EsiDate_t data structure
function 155	CICS_VerifyPassword	definition 146
CICS_VerifyPassword	function 152	CICS_EsiDetails_t data structure
function 151	CICS_ESI_ERR_PEM_NOT_ACTIVE	definition 148
CICS_ESI_ERR_NO_CICS return	return code	CICS_EsiTime_t data structure
code	CICS_ChangePassword	definition 147
CICS_ChangePassword	function 155	CICS_EXIT_BAD_ANCHOR return
function 154	CICS_VerifyPassword	code
CICS_SetDefaultSecurity	function 152	CICS_EciDataReturnExit 182
function 157	CICS_ESI_ERR_RESOURCE_SHORTAGE	CICS_EciDataSendExit 181
CICS_VerifyPassword	return code	CICS_EciExternalCallExit1 176
function 151	CICS_ChangePassword	CICS_EciExternalCallExit2 178
CICS_ESI_ERR_NO_SESSIONS	function 154	CICS_EciSetProgramAliasExit 183
return code	CICS_VerifyPassword	CICS_EciSystemIdExit 179
CICS_ChangePassword	function 151	CICS_EciTerminateExit 175
function 154	CICS_ESI_ERR_SECURITY_ERROR	CICS_EpiAddTerminalExit 189
CICS_VerifyPassword	return code	CICS_EpiDelTerminalExit 195
function 151	CICS_ChangePassword	CICS_EpiGetEventExit 196
CICS_ESI_ERR_NULL_	function 155	CICS_EpiReplyExit 194
NEWPASSWORD return code	CICS_VerifyPassword	CICS_EpiStartTranExit 193
CICS_ChangePassword	function 152	CICS_EpiSystemIdExit 198
function 155	CICS_ESI_ERR_SYSTEM_ERROR	CICS_EpiTermIdExit 191
CICS_ESI_ERR_NULL_	return code	CICS_EpiTermIdInfoExit 192
OLDPASSWORD return code	CICS_ChangePassword	CICS_EpiTerminateExit 187
CICS_ChangePassword	function 154	CICS_EpiTranFailedExit 199
function 155		

CICS_EXIT_BAD_PARM return code
 CICS_EciDataReturnExit 182
 CICS_EciDataSendExit 181
 CICS_EciExternalCallExit1 176
 CICS_EciExternalCallExit2 178
 CICS_EciSetProgramAliasExit 183
 CICS_EciSystemIdExit 180
 CICS_EpiAddTerminalExit 189
 CICS_EpiDelTerminalExit 195
 CICS_EpiGetEventExit 196
 CICS_EpiReplyExit 194
 CICS_EpiStartTranExit 193
 CICS_EpiSystemIdExit 198
 CICS_EpiTermIdExit 191
 CICS_EpiTermIdInfoExit 192
 CICS_EpiTranFailedExit 200

CICS_EXIT_BAD_STORAGE return code
 CICS_EciTerminateExit 175
 CICS_EpiTerminateExit 187

CICS_EXIT_CANT_INIT_EXITS return code
 CICS_EciInitializeExit 174
 CICS_EpiInitializeExit 186

CICS_EXIT_DONT_ADD_TERMINAL return code
 CICS_EpiAddTerminalExit 189
 CICS_EpiSystemIdExit 198

CICS_EXIT_GIVE_UP return code
 CICS_EciSystemIdExit 180

CICS_EXIT_NO_EXIT return code
 CICS_EciInitializeExit 174
 CICS_EpiInitializeExit 186

CICS_EXIT_OK return code
 CICS_EciDataReturnExit 182
 CICS_EciDataSendExit 181
 CICS_EciExternalCallExit1 176
 CICS_EciExternalCallExit2 178
 CICS_EciInitializeExit 174
 CICS_EciSetProgramAliasExit 183
 CICS_EciSystemIdExit 179
 CICS_EciTerminateExit 175
 CICS_EpiAddTerminalExit 189
 CICS_EpiDelTerminalExit 195
 CICS_EpiGetEventExit 196
 CICS_EpiInitializeExit 186
 CICS_EpiReplyExit 194
 CICS_EpiStartTranExit 193
 CICS_EpiSystemIdExit 198
 CICS_EpiTermIdExit 191
 CICS_EpiTermIdInfoExit 192
 CICS_EpiTerminateExit 187
 CICS_EpiTranFailedExit 199

CICS_ExternalCall 8, 55

CICS_SetDefaultSecurity function
 definition 156

CICS_VerifyPassword function
 definition 150

CICS3270.CBL copybook 121

CICS3270.H header file 121

CICS3270.INC include file 121

CicsClientStatus 54

cicseciexit 169

cicsepixexit 169

CICSEPIEXITINIT function 201

cicsepix.dll 201

CICSPRNT 201

CicsServerStatus 54

CICSTERM 201

client 1

code page 21, 29, 45, 50

ConnectionType 54

D

data conversion 21, 29, 45, 50

Data parameter
 CICS_EpiReply function 104
 CICS_EpiStartTran function 62, 101

DBCS 58

DCE authorization
 ECI 13
 EPI 67

DCE credentials
 ECI 13
 EPI 67

Description
 CICS_EciListSystems 55

Details parameter
 CICS_ChangePassword function 154
 CICS_EpiAddExTerminal function 93
 CICS_EpiAddTerminal function 88
 CICS_VerifyPassword function 151

DevType parameter
 CICS_EpiAddExTerminal function 92
 CICS_EpiAddTerminal function 87

DFHCNV macro 21, 29, 45, 50

distributed program link 4, 7

distributed transaction processing 4

E

ECI 3, 7

eci_abend_code
 field in ECI parameter block 16

eci_abend_code (*continued*)
 with ECI_SYNC call type 20

ECI_ASYNC call type
 definition 27

ECI_ASYNC_NOTIFY_MSG call type
 definition 159
 use 162, 163

ECI_ASYNC_NOTIFY_SEM call type
 definition 160
 use 162

eci_async_notify.sem_handle 160, 161, 162

eci_async_notify.win_fields.hinstance 160

eci_async_notify.win_fields.hwnd 160

eci_async_notify.window_handle 160

ECI_BACKOUT 19, 22, 28, 31

eci_call_type 8, 16
 field in ECI parameter block 16
 with ECI_ASYNC call type 28
 with ECI_GET_REPLY call type 44
 with ECI_GET_REPLY_WAIT call type 48
 with ECI_GET_SPECIFIC_REPLY call type 49
 with
 ECI_GET_SPECIFIC_REPLY_WAIT call type 53
 with ECI_STATE_ASYNC call type 40
 with ECI_STATE_SYNC call type 35
 with ECI_SYNC call type 19

eci_callback 27, 39
 field in ECI parameter block 17
 with ECI_ASYNC call type 32
 with ECI_STATE_ASYNC call type 42

ECI_CANCEL 163

ECI_CLIENTSTATE_INAPPLICABLE 54

ECI_CLIENTSTATE_UNKNOWN 54

ECI_CLIENTSTATE_UP 54

eci_commarea
 field in ECI parameter block 16
 with ECI_ASYNC call type 29
 with ECI_GET_REPLY call type 44
 with ECI_GET_SPECIFIC_REPLY call type 49
 with ECI_STATE_ASYNC call type 40
 with ECI_STATE_SYNC call type 35
 with ECI_SYNC call type 20

eci_commmarea_length
 field in ECI parameter block 16
 with ECI_ASYNC call type 29
 with ECI_GET_REPLY call
 type 45
 with ECI_GET_SPECIFIC_REPLY
 call type 50
 with ECI_STATE_ASYNC call
 type 40
 with ECI_STATE_SYNC call
 type 35
 with ECI_SYNC call type 21
 ECI_COMMIT 19, 22, 28, 30
 ECI_CONNECTED_NOWHERE 54
 ECI_CONNECTED_TO_CLIENT 54
 ECI_CONNECTED_TO_SEVER 54
 ECI_ERR_ALREADY_ACTIVE 25,
 33
 ECI_ERR_CALL_FROM
 _CALLBACK 18, 56
 ECI_ERR_CICS_DIED 24, 46, 51
 ECI_ERR_INVALID_CALL_TYPE 18
 ECI_ERR_INVALID_DATA
 _LENGTH 24, 33, 37, 42, 46, 50,
 56
 ECI_ERR_INVALID_DATA_AREA 25,
 34, 38, 43, 47, 51
 ECI_ERR_INVALID_EXTEND
 _MODE 24, 33, 37, 42
 ECI_ERR_INVALID_TRANSID 25,
 34
 ECI_ERR_INVALID_VERSION 18
 ECI_ERR_LUW_TOKEN 25, 33, 37,
 43
 ECI_ERR_MAX_SESSIONS 25, 47,
 52
 ECI_ERR_MAX_SYSTEMS 25, 47,
 52
 ECI_ERR_MORE_SYSTEMS 56
 ECI_ERR_NO_CICS 24, 33, 46, 51,
 56
 ECI_ERR_NO_REPLY 46, 51
 ECI_ERR_NO_SESSIONS 25, 34
 ECI_ERR_NO_SYSTEMS 56
 ECI_ERR_NULL_MESSAGE_ID 165
 ECI_ERR_NULL_SEM_HANDLE 165
 ECI_ERR_NULL_WIN_HANDLE 164
 ECI_ERR_REQUEST_TIMEOUT 18,
 165
 ECI_ERR_RESOURCE_SHORTAGE 25,
 33, 46, 51
 ECI_ERR_RESPONSE_TIMEOUT 165
 ECI_ERR_ROLLEDBACK 25, 47, 52
 ECI_ERR_SECURITY_ERROR 26,
 47, 52
 ECI_ERR_SYSTEM_ERROR 18, 56
 ECI_ERR_THREAD_CREATE
 _ERROR 33, 46, 51
 ECI_ERR_TRANSACTION
 _ABEND 25, 46
 ECI_ERR_TRANSACTION_ABEND 51
 ECI_ERR_UNKNOWN_SERVER 25,
 38, 47, 52
 ECI exits 171
 eci_extend_mode 10, 14, 19, 20, 24,
 28, 29, 33, 35, 36, 40, 41, 44, 49
 field in ECI parameter block 17
 with ECI_ASYNC call type 30
 with ECI_STATE_ASYNC call
 type 40
 with ECI_STATE_SYNC call
 type 36
 with ECI_SYNC call type 21
 ECI_EXTENDED 22, 30
 ECI_GET_REPLY call type
 definition 44
 use 160, 161
 ECI_GET_REPLY_WAIT call type
 definition 48
 ECI_GET_SPECIFIC_REPLY call type
 definition 49
 use 160, 161
 ECI_GET_SPECIFIC_REPLY_WAIT
 call type
 definition 53
 eci_luw_token 10, 11, 14
 field in ECI parameter block 17
 with ECI_ASYNC call type 31
 with ECI_STATE_ASYNC call
 type 41
 with ECI_STATE_SYNC call
 type 36
 with ECI_SYNC call type 22
 eci_message_id 160, 163
 eci_message_qualifier 9
 field in ECI parameter block 17
 with ECI_ASYNC call type 27,
 31
 with ECI_ASYNC_NOTIFY_MSG
 call type 160
 with ECI_ASYNC_NOTIFY_SEM
 call type 160
 with ECI_GET_SPECIFIC_REPLY
 call type 49, 50
 with ECI_STATE_ASYNC call
 type 39, 41
 with ECI_STATE_ASYNC_MSG
 call type 160
 with ECI_STATE_ASYNC_SEM
 call type 161
 ECI_NO_ERROR 24, 33, 37, 42, 45,
 50, 56
 ECI_NO_EXTEND 21, 30
 ECI parameter block 8, 16, 129, 162
 eci_password 20, 23, 24, 28, 32
 field in ECI parameter block 16
 with ECI_ASYNC call type 28
 with ECI_SYNC call type 20
 eci_password2 20, 28
 field in ECI parameter block 17
 with ECI_ASYNC call type 32
 with ECI_SYNC call type 23
 eci_program_name 10
 field in ECI parameter block 16
 with ECI_ASYNC call type 28
 with ECI_SYNC call type 19
 ECI_SERVERSTATE_DOWN 54
 ECI_SERVERSTATE_UNKNOWN 54
 ECI_SERVERSTATE_UP 54
 ECI_STATE_ASYNC call type
 definition 39
 ECI_STATE_ASYNC_MSG call type
 definition 160
 use 162, 163
 ECI_STATE_ASYNC_SEM call type
 definition 161
 use 162
 ECI_STATE_CANCEL 35, 36, 40,
 41, 44, 49
 ECI_STATE_CHANGED 36, 41
 ECI_STATE_IMMEDIATE 36, 40
 ECI_STATE_SYNC call type
 definition 35
 use 163
 ECI_STATUS 54
 ECI status block 13, 54
 ECI_SYNC call type
 definition 19
 use 163
 eci_sys_return_code 18
 eci_sysid
 field in ECI parameter block 17
 with ECI_ASYNC call type 31
 with ECI_GET_REPLY call
 type 45
 with ECI_GET_SPECIFIC_REPLY
 call type 50
 with ECI_STATE_ASYNC call
 type 41
 with ECI_STATE_SYNC call
 type 37
 with ECI_SYNC call type 22
 eci_system_name
 field in ECI parameter block 17
 with ECI_ASYNC call type 31

- eci_system_name *(continued)*
 - with ECI_STATE_ASYNC call type 42
 - with ECI_STATE_SYNC call type 37
 - with ECI_SYNC call type 23
- eci_timeout 163
 - field in ECI parameter block 17
 - with ECI_SYNC call type 21, 29
- eci_tpn
 - field in ECI parameter block 17
 - with ECI_ASYNC call type 32
 - with ECI_SYNC call type 24
- eci_transid
 - field in ECI parameter block 16
 - with ECI_ASYNC call type 29
 - with ECI_SYNC call type 20
- eci_userid 20, 23, 28, 32
 - field in ECI parameter block 16
 - with ECI_ASYNC call type 28
 - with ECI_SYNC call type 19
- eci_userid2 20, 28
 - field in ECI parameter block 17
 - with ECI_ASYNC call type 32
 - with ECI_SYNC call type 23
- eci_version 19
 - field in ECI parameter block 17
 - with ECI_ASYNC call type 31
 - with ECI_GET_REPLY call type 45
 - with ECI_GET_SPECIFIC_REPLY call type 50
 - with ECI_STATE_ASYNC call type 41
 - with ECI_STATE_SYNC call type 37
 - with ECI_SYNC call type 23
- ECI_VERSION_0 17, 23, 31, 37, 42, 45, 50
- ECI_VERSION_1 17, 23, 31, 37, 42, 45, 50
- ECI_VERSION_1A 17, 23, 24, 31, 32, 37, 42, 45, 50
- eci_window_handle 162
- EDF 58
- EPI 4, 57
 - 3270 data streams 121
 - 3270 order codes 123
 - constants 68
 - data structures 68
 - events 114
 - functions 77
 - Microsoft Windows considerations 125
- EPI exits 184
- ESI 5, 143
 - benefits 144
 - constants 145
 - data structures 145
 - functions 149
 - overview 143
- Event parameter
 - CICS_EpiGetEvent function 111
- EXEC CICS CONVERSE 65, 117
- EXEC CICS LINK 7
- EXEC CICS RECEIVE 65, 102, 104, 117, 121, 122, 123
- EXEC CICS RECEIVE BUFFER 117, 123
- EXEC CICS RECEIVE MAP 121, 122
- EXEC CICS RETURN
 - TRANSID option 65
- EXEC CICS SEND 65, 116, 121
- EXEC CICS SEND MAP 121
- EXEC CICS START
 - DELAY option 58
- extended calls 4
- F**
- front end programming interface 4
- H**
- hWnd parameter
 - CICS_EpiInitialize function 125
- L**
- List parameter 56
 - CICS_EciListSystems 55
 - CICS_EpiListSystems function 84
- LITLINK compiler directive 129, 132, 137, 139, 142
- load balancing 201
- logical unit of work 9
- M**
- MsgId parameter
 - CICS_EpiInitialize function 125
- N**
- Namespace parameter
 - CICS_EciListSystems 55
 - CICS_EpiAddTerminal function 86
 - CICS_EpiListSystems function 84
- NetName parameter
 - CICS_EpiAddExTerminal function 91
 - CICS_EpiAddTerminal function 87
- NewPassword parameter
 - CICS_ChangePassword function 153
- NotifyFn parameter
 - CICS_EpiAddExTerminal function 93
 - CICS_EpiAddTerminal function 63, 88
- O**
- OldPassword parameter
 - CICS_ChangePassword function 153
- P**
- password expiration management (PEM) 5, 143
- Password parameter
 - CICS_EpiSetSecurity function 99
 - CICS_SetDefaultSecurity function 156
 - CICS_VerifyPassword function 150
- PEM (password expiration management) 5, 143
- program link calls 9
- R**
- reserved1
 - field in ECI parameter block 17
 - with ECI_ASYNC call type 30
 - with ECI_STATE_ASYNC call type 40
 - with ECI_STATE_SYNC call type 35
 - with ECI_SYNC call type 21
- REXX 127, 129, 130, 131, 133
- S**
- screen size 57
- security considerations
 - ECI 12
 - EPI 66
- semaphore 160, 161
- SenseCode parameter
 - CICS_EpiSenseCode function 108
- server implementation 2
- signon capable 67
- signon incapable 66
- Size parameter
 - CICS_EpiReply function 104
 - CICS_EpiStartTran function 102
- structured fields 58
- synchronous calls 3

SysErr parameter
 CICS_EpiGetSysError
 function 112

system information structure 55, 84,
 130

System parameter
 CICS_ChangePassword
 function 154
 CICS_EpiAddExTerminal
 function 91
 CICS_EpiAddTerminal
 function 86
 CICS_SetDefaultSecurity
 function 156
 CICS_VerifyPassword
 function 150

SystemName parameter
 CICS_EciListSystems 55

Systems parameter 56
 CICS_EciListSystems 55
 CICS_EpiListSystems
 function 84

T

terminal, signon capable 67
 terminal, signon incapable 66
 terminal index 59, 86, 91

TermIndex parameter
 CICS_EpiAddExTerminal
 function 93
 CICS_EpiAddTerminal
 function 88
 CICS_EpiATISState function 106
 CICS_EpiDelTerminal
 function 97
 CICS_EpiGetEvent function 110
 CICS_EpiGetSysError
 function 112
 CICS_EpiInquireSystem
 function 96
 CICS_EpiPurgeTerminal
 function 98
 CICS_EpiReply function 104
 CICS_EpiSenseCode
 function 108
 CICS_EpiSetSecurity function 99
 CICS_EpiStartTran function 101

time-out 161

TransId parameter
 CICS_EpiStartTran function 62,
 101

U

user-defined
 CICS_EpiSystemIdExit 198

user-defined (*continued*)
 CICS_EpiTerminateExit 187

user-defined return code
 CICS_EciDataReturnExit 182
 CICS_EciDataSendExit 181
 CICS_EciExternalCallExit1 176
 CICS_EciExternalCallExit2 178
 CICS_EciInitializeExit 174
 CICS_EciSetProgramAliasExit 183
 CICS_EciSystemIdExit 180
 CICS_EciTerminateExit 175
 CICS_EpiAddTerminalExit 189
 CICS_EpiDelTerminalExit 195
 CICS_EpiGetEventExit 196
 CICS_EpiInitializeExit 186
 CICS_EpiReplyExit 194
 CICS_EpiStartTranExit 193
 CICS_EpiTermIdExit 191
 CICS_EpiTermIdInfoExit 192
 CICS_EpiTranFailedExit 200

UserId parameter

CICS_ChangePassword
 function 153
 CICS_EpiSetSecurity function 99
 CICS_SetDefaultSecurity
 function 156
 CICS_VerifyPassword
 function 150

V

Version parameter

CICS_EpiInitialize function 77,
 82

W

Wait parameter

CICS_EpiGetEvent 63
 CICS_EpiGetEvent function 110

WorkLoad Manager 183



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC33-1435-04

