CICS Transaction Server for z/OS
Version 5 Release 2

Customization Guide

SC34-7269-00

CICS Transaction Server for z/OS
Version 5 Release 2

Customization Guide

SC34-7269-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 921.

This edition applies to the IBM CICS Transaction Server for z/OS Version 5 Release 2 (product number 5655-Y04) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Preface

What this manual is about

This manual documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM® CICS® Transaction Server Version 5 Release 2.

This manual provides the information needed to extend and modify an IBM CICS Transaction Server for z/OS® system to match your requirements. It describes how you can tailor your system by coding exit programs, by replacing specific CICS-supplied default programs with versions that you write yourself, and by adapting sample programs.

Who this manual is for

This manual is for those responsible for extending and enhancing a CICS system to meet the special processing needs of an installation.

What you need to know to understand this manual

To use the information in this manual you must be familiar with some of the architecture of CICS and the programming interface to CICS. Programming interface information is given in the *CICS Application Programming Reference* and *CICS System Programming Reference*.

Resource definition information is in the *CICS Resource Definition Guide*

To use the following sections you must be familiar with the IBM ACF/VTAM telecommunications access method:

- Chapter 11, "Writing a node error program," on page 465
- Chapter 12, "Writing a program to control autoinstall of LUs," on page 499
- Chapter 14, "Writing a program to control autoinstall of APPC connections," on page 527

If your task involves error processing, you might need to consult *CICS Messages and Codes Vol 1*, *CICS Problem Determination Guide*, or the *CICS Diagnosis Reference*.

How to use this manual

The manual is divided into parts according to subject matter. Each part is made up of one or more major sections. Major section are usually further subdivided into topics. The parts and major sections of the book are self-contained. Use an individual part or section as a guide when performing the task described in it.

Location of topics in the information center

The topics in this publication can also be found in the CICS information center. The information center uses content types to structure how the information is displayed.

The information center content types are generally task-oriented, for example; upgrading, configuring, and installing. Other content types include reference, overview and scenario or tutorial-based information. The following mapping shows the relationship between topics in this publication and the information center content types, with links to the external information center:

Table 1. Mapping of PDF topics to information center content types. This table lists the relationship between topics in the PDF and topics in the content types in the information center

Set of topics in this publication	Location in the information center
• Parts 1 to 6 and part 8 • Part 7 • Appendixes	• Developing system programs • Securing • System programming reference in Reference

Syntax notation and conventions used in this manual

The symbols { }, [ ], and | are used in the syntax descriptions of the EXEC CICS commands and macros referred to in this manual. They are not part of the command and you should not include them in your code. Their meanings are as follows:

- Braces { } enclose two or more alternatives, one of which you **must** code.
- Square brackets [ ] tell you that the enclosed is optional.
- The “or” symbol | separates alternatives.

In addition to these symbols, the following conventions apply:

- Punctuation symbols and uppercase characters should be coded exactly as shown.
- Lowercase characters indicate that user text should be coded as required.
- Default values are shown like this: DEFAULT.
- Options that are enclosed neither in braces { } nor in square brackets [ ] are mandatory.
- The ellipsis ... means that the immediately preceding option can be coded one or more times.
- All EXEC CICS commands require a delimiter appropriate to the language of the application. For a COBOL program this is ‘END-EXEC', for example. Delimiters are not included in the syntax descriptions of the commands.

Changes in CICS Transaction Server for z/OS, Version 5 Release 2

For information about changes that have been made in this release, please refer to *What's New* in the information center, or the following publications:

- *CICS Transaction Server for z/OS What's New*
- *CICS Transaction Server for z/OS Upgrading from CICS TS Version 5.1*
- *CICS Transaction Server for z/OS Upgrading from CICS TS Version 4.2*
- *CICS Transaction Server for z/OS Upgrading from CICS TS Version 4.1*
- *CICS Transaction Server for z/OS Upgrading from CICS TS Version 3.2*
- *CICS Transaction Server for z/OS Upgrading from CICS TS Version 3.1*

Any technical changes that are made to the text after release are indicated by a vertical bar (|) to the left of each new or changed line of information.

Part 1. Customizing with user exit programs

CICS provides a number of *exit points* at which it can transfer control to a program that you have written (a *user exit program*). You can use exit programs to extend or modify the way CICS operates.

Chapter 1. Global user exit programs

You can use the CICS *global user exit points* with programs of a special type that you write yourself called *global user exit programs* to customize your CICS regions.

A global user exit point or *global user exit* is a place in a CICS module or domain at which CICS can transfer control to a global user exit program that you have written. CICS can resume control when your exit program has finished its work.

Each global user exit point has a unique identifier, and is located at a point in the module or domain at which it might be useful to do some extra processing. For example, at exit point XSTOUT in the statistics domain, an exit program can be given control before each statistics record is written to the SMF data set, and can access the relevant statistics record. You might want to use an exit program at this exit point to examine the statistics record and suppress the writing of unwanted records.

Global user exit support is provided automatically by CICS. However, there are several conventions that govern how you write your exit program. These conventions are described in Writing global user exit programs. Because global user exit programs work as if they were part of the CICS module or domain, there are limits on the use you can make of CICS services. Most global user exit programs cannot use **EXEC CICS** commands, but can call some CICS services with the exit programming interface (XPI). For more information, see Using CICS services.

Note: The source and object compatibility of CICS management modules are not guaranteed from release to release. Any changes that affect exit programs are documented in the upgrading documentation.

Writing global user exit programs

You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Remember: A reentrant program is coded to allow one copy of itself to be used concurrently by several tasks; it does not modify itself while running. A quasireentrant program is serially reusable by different tasks. When it receives control it must be in the same state as when it relinquished control. Such a program can modify itself while running, and is therefore not fully reentrant.

For more information about quasireentrant programs, see Multithreading: Reentrant, quasi-reentrant, and threadsafe programs in Developing applications.

Register conventions

Register values are provided on entry to an exit program and only certain register values are guaranteed.

The register values that you can use on entry to an exit program are as follows:

- Register 1 contains the address of the user exit parameter list DFHUEPAR.

Write-to-operator (WTO) commands use register 1. If your exit program uses WTO commands, you should save the address of DFHUEPAR first.

- Register 13 contains the address of the standard register save area where your exit program should store its own registers immediately after being invoked. This address is also in the field UEPEPSA in the parameter list pointed to by register 1.

  If you want to issue operating system requests that use register 13 to point to a save area, you must switch register 13 to point to another save area. You must restore register 13 to its original contents before returning from your user exit program to the caller.
- Register 14 contains the return address to which the exit program should branch on completion of its work. You do this using the BR 14 instruction after restoring the calling module's registers, or using the RETURN macro.
- Register 15 contains the entry address of the exit program.

The exit program must save and restore any registers that it modifies, using the save area addressed by register 13.

31-bit addressing implications

The following lists show you the CICS addressing and access register implications.

- The global user exit is started in 31-bit AMODE.
- The global user exit can be either RMODE 24 or RMODE ANY.
- If you find it necessary to switch to 24-bit AMODE in the exit program, be sure to return correctly in 31-bit AMODE.
- Ensure that the exit program is in 31-bit AMODE for XPI calls.
- Some of the parameters passed in DFHUEPAR are addresses of 31-bit storage (above the 16 MB line).
- The global work area for the exit program can be in 24-bit storage or 31-bit storage. When you define the exit, use the GALOCATION option on the ENABLE PROGRAM command to specify the location of the storage. You can use the EXTRACT EXIT command to check the address of a global work area for an exit program.

Access register implications

- The global user exit is started in primary-space translation mode. For information about translation modes, see the *IBM z/Architecture Principles of Operation* manual.
- The contents of the access registers are unpredictable. For information about access registers, see the *IBM z/Architecture Principles of Operation* manual.
- If the global user exit modifies any access registers, it must restore them before returning control. CICS does not provide a save area for this purpose.
- The global user exit must return control in primary addressing mode.

Using CICS services

The rules governing the use of CICS services in exit programs vary, depending on the exit point from which the exit program is being started.

About this task

The following general rules apply:

- No CICS services can be started from any exit point in the dispatcher domain.

- CICS services can be started by using the exit programming interface (XPI) from most exits. If you use the XPI, note the rules and restrictions that are listed for each exit and each of the XPI macros. The XPI is described in The user exit programming interface (XPI).
- Some CICS services can be requested by using EXEC CICS commands from some exits. The valid commands are listed in the detailed descriptions of the exits. If no commands are listed, it means that no EXEC CICS API or SPI commands are supported. EXEC CICS commands that cause an XCTL (either directly or implied); for example, **EXEC CICS XCTL** or **EXEC CICS SHUTDOWN** must never be used.

  An exit program started at an exit that does not support the use of EXEC CICS commands must not call a task-related user exit program (TRUE). Calling a TRUE is equivalent to issuing an EXEC CICS command. Exceptions to this rule are programs started from the XFCFRIN and XFCFROUT exits, which can call a TRUE. TRUEs are described in Task-related user exit programs.

  Note: In exits which support the use of EXEC CICS file control commands, file commands that form a related sequence (such as **EXEC CICS STARTBR**, **EXEC CICS READNEXT**, and **EXEC CICS ENDBR**) must all be issued in the same invocation of the exit program.

  For example, if one invocation of an exit program issues an **EXEC CICS STARTBR** command, and the next invocation of the exit program for that same task issues an **EXEC CICS READNEXT** command, the READNEXT fails with an INVREQ condition.
- All exit programs that issue EXEC CICS commands must first address the EIB. This is not done automatically by using the DFHEIENT macro, as is the case with normal EXEC assembler language programs. Therefore, the first EXEC command to be issued from an exit program must be EXEC CICS ADDRESS EIB (eib-register), where "eib-register" is the default register (R11) or the register given as a parameter to the DFHEIENT macro.

  All exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See "Returning values to CICS" on page 10.

Note:

- If your global user exit program does not contain EXEC CICS commands, do not use the CICS command-level translator when assembling the program.
- Do not make non-CICS (for example, RACF® or MVS™) system service calls from global user exit programs.
- If an operating system request causes a wait, your whole CICS system stops until the operating system request has been serviced.

Using EXEC CICS and XPI calls in the same exit program

There are a number of exits where you can use both EXEC CICS commands and XPI calls, but you should ensure that there is no conflict in the usage of register 13.

To avoid such conflict, use the DATAREG option on the DFHEIENT macro (see "XPI register usage" on page 386 for information).

For an example of how to use EXEC CICS commands and XPI calls in the same global user exit program, see Appendix G, "The example program for the XTSEREQ global user exit, DFH$XTSE," on page 917

Using channels and containers

Global user exit programs can access channels and containers created by application programs. They can also create their own channels and pass them to programs which they call.

For information about channels and containers, see the *CICS Application Programming Guide*.

Assembler programs and LEASM

Assembler programs translated with the LEASM option cannot be used as global user exit programs.

LEASM is used to produce Language Environment® conforming main programs in assembler. For information about the LEASM translator option, see Translator options in Developing applications.

EDF and global user exits

If you use the Execution Diagnostic Facility (EDF) to debug your applications, you must take care when compiling exit programs that issue EXEC CICS commands.

Normally, if an exit program issues EXEC CICS commands, these are displayed by EDF, if the latter is active. They appear between the "Start of Command" and "End of Command" screens for the command that caused the exit to be driven. If you want to suppress the display of EXEC CICS commands issued by your exit program, you must specify the NOEDF option when you translate the program. You should always specify NOEDF for programs in a production environment.

If an exit program that may be invoked during recovery processing issues EXEC CICS commands, you must translate it with the NOEDF option. Failure to do so may cause EDF to abend.

The global work area

When you enable an exit program, you can ask CICS to provide a global work area for the exit program. An exit program can have its own global work area, or it can share a work area that is owned by another exit program.

When you issue the **ENABLE PROGRAM** command to define the exit, you can specify the GALENGTH and GALOCATION options to make CICS provide a global work area for the exit program. GALENGTH specifies the length of the global work area in bytes, and GALOCATION specifies whether it is in 24-bit storage or 31-bit storage. Alternatively, you can specify the GAENTRYNAME option to name another currently enabled user exit, and your exit program shares the global work area that CICS has already provided for the named exit.

The global work area is associated with the exit program rather than with the exit point. For ease of problem determination, the global work area should be shared only by exit programs that are invoked from the same management module or domain. The address and length of the global work area are addressed by parameters **UEPGAA** and **UEPGAL** of the DFHUEPAR parameter list, which is described in "DFHUEPAR standard parameters" on page 8. If a user exit program does not own a global work area, UEPGAA is set to zero.

Application programs can communicate with user exit programs that use or share the same global work area. The application program uses the **EXEC CICS EXTRACT EXIT** command to obtain the address and length of the global work area.

A work area is freed only when all of the exit programs that use it are disabled. For examples of how to use a global work area, see the sample global user exit programs, which are listed in "Sample global user exit programs" on page 19.

Making trace entries

If tracing is active, you can specify that an entry in the CICS trace table is made immediately before and immediately after the exit program runs.

About this task

Procedure

- Use either of the following methods to create a trace entry before and after the exit program runs:
  - The UE option of the CETR transaction.
  - The UE option of the **EXEC CICS SET TRACETYPE** command.
- If your global user exit is in a domain, you can add extra trace calls to provide additional diagnostic information by setting the AP option of **EXEC CICS SET TRACETYPE** to level 1 or 2.
- Depending on which exit point you are using, you might be able to use the XPI DFHTRPTX TRACE_PUT macro to create trace entries in the user exit program. This macro is described in Chapter 3, "The user exit programming interface (XPI)," on page 379. The individual descriptions of the global user exit points indicate whether you can use the XPI DFHTRPTX macro.

Parameters passed to the global user exit program

The address of a parameter list is passed to the user exit program in register 1. The list contains some standard parameters that are passed to all global user exit programs, and might also contain some exit-specific parameters that are unique to the exit point from which the exit program is being invoked.

The exit-specific parameters are described with the individual exits in the section "Global user exit points (in alphabetical order)" on page 31. The standard parameter list is described in the following section.

You can map the parameter list using the DSECT DFHUEPAR, which is generated by the macro instruction

```
DFHUEXIT TYPE=EP,ID=exit_point_identifier
```

The ID parameter provides the extra data definitions that you need to map any exit-specific parameters. For example, the macro instruction

```
DFHUEXIT TYPE=EP,ID=XTDIN
```

generates the DSECT to map the standard parameters followed by the parameters that are specific to exit point XTDIN in the transient data program. If your exit program is to be invoked at more than one exit point, you can code up to 256 characters of relevant exit identifiers on a single DFHUEXIT macro instruction. For example:

```
DFHUEXIT TYPE=EP,ID=(XMNOUT,XSTOUT,XTDIN)
```

If your exit program is to be invoked at every global user exit point, you can code:

```
DFHUEXIT TYPE=EP,ID=ALL
```

If your user exit program is to be used both as a global user exit program and as a task-related user exit program, you must code both:

```
DFHUEXIT TYPE=EP,ID=exit_point_identifier
```

and:

```
DFHUEXIT TYPE=RM
```

(in this order) to generate the DSECTs appropriate to both types of user exit.

If a global user exit program needs to use the DFHRMCAL macro to invoke an external RMI, the DFHRMCAL macro instruction must follow the DFHUEXIT macro.

DFHUEPAR standard parameters

The DFHUEPAR standard parameters are passed to all global user exit programs.

```
DFHUEPAR DSECT
*  STANDARD PARAMETERS
UEPEXN   DS    A      ADDRESS OF EXIT NUMBER
UEPGAA   DS    A      ADDRESS OF GLOBAL WORK AREA
*                      (ZERO = NO WORK AREA)
UEPGAL   DS    A      ADDRESS OF GLOBAL WORK AREA LENGTH
UEPCRCA  DS    A      ADDRESS OF CURRENT RETURN-CODE
UEPTCA   DS    A      RESERVED
UEPCSA   DS    A      RESERVED
UEPEPSA  DS    A      ADDRESS OF REGISTER SAVE AREA
*                      FOR USE BY EXIT PROGRAM
UEPHMSA  DS    A      ADDRESS OF SAVE AREA USED FOR
*                      HOST MODULE'S REGISTERS
UEPGIND  DS    A      ADDRESS OF CALLER'S TASK INDICATORS
UEPSTACK DS    A      ADDRESS OF KERNEL STACK ENTRY
UEPXSTOR DS    A      ADDRESS OF STORAGE FOR XPI PARAMETERS
UEPTRACE DS    A      ADDRESS OF TRACE FLAG
```

UEPEXN
Points to a 1-byte binary field, the contents of which identify the global user exit point from which the exit program is called. You require this information if your exit program can be called from more than one exit point.

DFHUEXIT TYPE=EP generates a list of equated values that relate the exit names (exitids) to the exit numbers used internally by CICS to identify the exits. Always use the exitids, because the exit numbers might change in any future releases of CICS.

UEPGAA
Points to the global work area that was provided for the exit program when it was enabled. This parameter is set to zero if no global work area is provided.

UEPGAL
Points to a halfword that contains the length of the global work area.

UEPCRCA
Points to a halfword that is to contain the return code value from the exit program. When more than one program is called at a user exit, this field contains (on entry to the second and subsequent programs) the return code that was set by the previously called program.

For an example of how an exit program can set a different return code from that returned by a previous exit program at the same exit point, see the code snippet in "Invoking more than one exit program at a single exit" on page 14.

UEPTCA
Points to fetch-protect storage. Use of this field results in an abend ASRD at run time.

UEPCSA
Points to fetch-protect storage. Use of this field results in an abend ASRD at run time.

UEPEPSA
Points to a save area in which the exit program stores its own registers on entry. When the exit program is entered, register 13 also points to this area. The convention is to save registers 14, 15, and 0 - 12 at offset 12 (decimal) onward.

UEPHMSA
Points to the save area containing the registers of the calling module. Values for registers 14, 15, and 0 - 13 are stored in this order from offset 12 (decimal) in this area.

Apart from register 15, which contains the return code value from the exit program, the values in this save area are used by CICS to reload the registers when returning to the calling CICS module. They must not be corrupted.

This address is *not* passed to global user exit programs called from exit points in CICS domains.

UEPGIND
Points to a 3-byte field containing indicators for use in AP domain user exits. For non-AP domain user exits, the indicators are always zero.

The first indicator byte can take one of two symbolic values, UEPGANY and UEPGCICS. You can test these values to determine whether data locations can be above or below 16 MB, and whether the application storage is in CICS-key or user-key storage:

UEPGANY
The application can accept addresses above 16 MB. If the symbolic value is not UEPGANY, the application can accept an address only below 16 MB.

UEPGCICS
The application working storage and the task-lifetime storage are in CICS-key storage (TASKDATAKEY=CICS). If the symbolic value is not UEPGCICS, the application working storage and the task-lifetime storage are in user-key storage (TASKDATAKEY=USER).

The second and third bytes contain a value indicating the TCB mode of the caller of the global user exit program. This value is represented in DFHUEPAR as both a 2-character code and a symbolic value, as follows:

Table 2. TCB indicators in DFHUEPAR

Symbolic value	2-byte code	Description
UEPTQR	QR	The quasi-reentrant mode TCB
UEPTRO	RO	The resource-owning mode TCB
UEPTCO	CO	The concurrent mode TCB

Table 2. TCB indicators in DFHUEPAR (continued)

Symbolic value	2-byte code	Description
UEPTSZ	SZ	The FEPI mode TCB
UEPTRP	RP	The ONC/RPC mode TCB
UEPTFO	FO	The file-owning mode TCB
UEPTSL	SL	The sockets listener mode TCB
UEPTSO	SO	The sockets mode TCB
UEPTS8	S8	The secure sockets layer mode TCB
UEPTD2	D2	The CICS DB2 housekeeping mode TCB
UEPTL8	L8	An L8 open TCB, used for OPENAPI TRUEs, or OPENAPI programs that are in CICS key
UEPTL9	L9	An L9 open TCB, used for OPENAPI programs that are in user key
UEPTEP	EP	Event processing TCB
UEPTTP	TP	The TP open TCB, used to own the Language Environment enclave and THRD TCB pool for a JVM server.
UEPTT8	T8	A T8 TCB, used by a JVM server to process multithreaded processing.
UEPTX8	X8	An X8 open TCB, used for C and C++ programs, compiled with the XPLINK option, that are in CICS key
UEPTX9	X9	An X9 open TCB, used for C and C++ programs, compiled with the XPLINK option, that are in user key

UEPSTACK
Points to the kernel stack entry. This value must be moved to register 13 of the exit program before calling the XPI. For more information, see Chapter 3, "The user exit programming interface (XPI)," on page 379. The storage addressed by this field must not be altered. If it is corrupted, your exit program will have unpredictable effects on your CICS system.

UEPXSTOR
Points to a 1024-byte area of DFHUEH-owned LIFO storage that the exit program uses when calling the XPI. For more information, see Chapter 3, "The user exit programming interface (XPI)," on page 379.

UEPTRACE
Points to the trace flag, which indicates whether tracing is on in the calling management module or domain. Use this parameter to control your use of the XPI TRACE_PUT macro in line with the tracing in the CICS module or domain. Use the XPI TRACE_PUT function only when tracing is on. The trace flag is a single byte, with a top bit set on when tracing is switched on. You test this setting using the symbolic value UEPTRON. The rest of the byte addressed by UEPTRACE is reserved, and its contents must not be corrupted.

Returning values to CICS

At some exit points, you can influence what CICS does on return from an exit program by supplying a return code value.

About this task

You must set the return code value in register 15 before leaving the exit program.

Procedure

- Use character string values rather than using hard-coded values. Character strings equating to valid return code values are provided with the parameter list for each exit point. For example, at exit XMNOUT in the monitor domain, you are presented with the address of a monitoring record. If you decide in your exit program that this record should not be written to SMF, you can set the return code value UERCBYP (meaning "bypass this record") before returning to CICS and CICS suppresses the record.
- If you have more than one exit program running for an exit point, use parameter **UEPCRCA** of DFHUEPAR to set the return code. For more information, see "Invoking more than one exit program at a single exit" on page 14
- If your exit program issues an EXEC CICS command and use the DFHEIENT macro, you must use this macro to set the return code. The DFHEIRET macro:
  - Restores registers
  - Places a return code in register 15 after the registers are restored
  - Returns control to the address in register 14.

  For example:

  ```
  DFHEIRET RCREG=nn
  ```

  where *nn* is the number of any register (other than 13) that contains the return code to be placed in register 15 after the registers are restored.

Results

If you supply a return code value that is not expected at a particular exit point, the default return code indicating a normal response (usually UERCNORM) is assumed, unless you set the return code UERCPURG. You are strongly advised not to let the return code default to the normal response as the result can be unpredictable. The normal response tells CICS to continue processing as if the exit program had not been invoked, and it is a valid option at most global user exit points. The exceptions are shown in the list of return codes provided with each exit description.

Restrictions on the use of fields as programming interfaces

Some CICS data area control block field definitions must not be used as part of a CICS application programming interface.

The *CICS Data Areas* contains definitions of the control block fields that form part of the Product-sensitive and General-use programming interfaces of CICS. Fields that are not defined in the CICS Data Areas manual as either Product-sensitive programming interface or General-use programming interface fields are not intended for your use as part of a CICS programming interface.

Exit programs and the CICS storage protection facility

When you are running CICS with the storage protection facility, this affects the execution key in which your user exit programs run and the storage key of data storage that your exit programs obtains.

Execution key for global user exit programs

When you are running with storage protection active, CICS always invokes global user exit programs in CICS key. Even if you specify EXECKEY(USER) on the program resource definition, CICS forces CICS key when it passes control to the

exit program. However, if a global user exit program itself passes control to another program (via a link or transfer-control command), the program thus invoked is executed according to the execution key (EXECKEY) defined in its program resource definition.

You are strongly recommended to specify EXECKEY(CICS) when defining both global user exit programs and programs to which an exit program passes control.

Data storage key for global user exit programs

The storage key of storage used by global user exit programs depends on how the storage is obtained:

- The CICS-supplied storage addressed by the **UEPXSTOR** parameter of DFHUEPAR, and any global work area specified when an exit program is enabled, are always in CICS key.
- Global user exit programs that can issue EXEC CICS commands can obtain storage by:
  - Explicit **EXEC CICS GETMAIN** commands
  - Implicit storage requests as a result of EXEC CICS commands that use the SET option.

  The default storage key for storage obtained by EXEC CICS commands is set by the TASKDATAKEY of the transaction under which the exit program is invoked.

  As an example, consider a transaction defined with TASKDATAKEY(USER) that issues a file control request, which causes an XFCREQ global user exit program to be invoked. In this case, any implicit or explicit storage acquired by the exit program by means of an EXEC CICS command is, by default, in user-key storage. However, on an EXEC CICS GETMAIN command, the exit program can override the TASKDATAKEY option by specifying either CICSDATAKEY or USERDATAKEY.
- When an exit program obtains storage using an XPI GETMAIN call, the storage key depends on the value specified on the STORAGE_CLASS option, which is mandatory, and which overrides the value of TASKDATAKEY.

Exit programs and transaction isolation

When you are running CICS with the transaction isolation facility (TRANISO=YES), the exit program will inherit the subspace of the application that caused the exit to be invoked.

If your GLUE needs to access storage belonging to a task other than the invoking task, it should use the **DFHSMSRX SWITCH_SUBSPACE XPI** command to switch to base space. It does not need to switch back to subspace mode before returning to CICS as CICS will restore the subspace mode if necessary.

Errors in user exit programs

Because global user exit programs are an extension to CICS code, they are subject to the environment that CICS is running in when they are called.

If an error is detected at an exit point, CICS issues messages indicating which exit program was in error, the place in the program at which the error occurred, and the name of the associated exit point. The detection of an error is not guaranteed, because it depends on the CICS environment at the time of error, and on the nature of the error. For example, CICS might not recognize a looping user exit program, since the detection mechanism may have been turned off. Also, an abend

in one of the exits XPCABND, XPCTA, or XSRAB may cause CICS to terminate abnormally, because an abend during abend processing causes CICS to terminate.

Exit programs invoked at some exit points (for example, XTSEREQ, XTSEREQC, XICEREQ, XICEREQC, XTDEREQ, or XTDEREQC) can enter a loop by issuing a recursive command (such as a TS command at exit point XTSEREQ). The exits most likely to be affected provide a recursion count parameter, UEPRECUR, that you can use to prevent such loops.

Important

When coding user exit programs, you should bear in mind that the code is executed as an extension of CICS code, rather than as a transaction, and any errors could have disastrous results.

Defining, enabling, and disabling an exit program

When you have written an exit program, you must define it to CICS. You can enable your exit program using an exit point or the **EXEC CICS ENABLE** command. You can disable your exit program using the **EXEC CICS DISABLE** command.

Procedure

1. Define the exit program using the **CEDA DEFINE PROGRAM** command, specifying RELOAD(NO).
2. Install the exit program.
3. Enable the exit program using one of the following methods:
   - If your exit program is using an exit point in the user log record recovery program or the file control recovery control program, you can use the **TBEXITS** system initialization parameter.
   - Otherwise, use the **EXEC CICS ENABLE** command to enable the user exit program.

   If you enable a global user exit program before it has been installed and LPA=YES is specified as a system initialization parameter, CICS scans the LPA for the program. If message DFHLD0109I is issued, it means that CICS was unable to find the program in the LPA and is using the version in DFHRPL or a dynamic LIBRARY.
4. When you have finished using the exit program, you can disable it using the **EXEC CICS DISABLE** command.

Example

For examples of how to enable and disable global user exit programs, see the sample programs listed in “Sample global user exit programs” on page 19.

Viewing active global user exits

You can use the Web User Interface to view all of the global user exit programs that are running in your CICS regions.

Before you begin

You must have CICSPlex SM installed and configured to perform this task.

About this task

Procedure

1. Log on to the Web User Interface.
2. Select **Operations views** > **Exit operations views** > **Global user exits**. The global user exits view displays details for all of the programs that are using global user exit points in your CICS regions.
3. Select the name of the global user exit program that you are interested in to view the details of the program definition.

What to do next

Using this view you can perform additional tasks such as enabling and disabling global user exit programs.

Invoking more than one exit program at a single exit

You can invoke more than one exit program from a single global user exit point.

Although such programs can work independently, you should note the following points:

- An exit program is only called at an exit if it has been made available for execution with the START option of the **EXEC CICS ENABLE** command. The order of invocation, when more than one exit program has been started at an exit point, is the order in which the programs were activated (that is, the order in which the **EXEC CICS ENABLE** commands associated them with the exit point). When programs work on the same data area, you should consider the order in which they are invoked. For example, in a terminal control output exit, an exit program might manipulate the same message in different ways, depending on the way an earlier exit program acted.
- Return code management is more complicated than it is for single programs. Each exit program sets a return code in register 15 as usual. The second and subsequent programs invoked from a single exit point can access the return code value set by the preceding program (the "current return code") using the parameter **UEPCRCA** of DFHUEPAR.

  The following rules apply to return codes if a number of user exit programs set return code values when invoked on a single exit:

  – If a user exit program supplies the same return code value as the previous program (addressed by UEPCRCA), then CICS acts on that value.
  – If a user exit program supplies a different return code value from the previous program (addressed by UEPCRCA), CICS ignores both values and resets the "current return code" to the default value, usually UERCNORM, before calling any further exit programs for that exit point.
  – If a user exit program sets an eligible value in register 15 and changes the "current value" field to match (as addressed by UEPCRCA), the new value is adopted and passed on to the next program (if any), or back to the calling CICS module or domain.

    The following code snippet shows how an exit program can set a different return code from the "current return code", returned by a previous exit program, and cause CICS to act on the new code.

```
            LA    R15,UERCTDOK  Set the contents of reg 15 to a value of 4
            L     R6,UEPCRCA    Set reg 6 to the address of the half word
*                               containing the current return code
            STH   R15,0(,6)     Store the new return code at the location
*                               of the current return code.
          . . .
```

Invoking a single exit program at more than one exit

To invoke a single exit program from more than one exit point, you must issue an **ENABLE** command for each of the exit points.

For programming information about how to issue an ENABLE command, see the *CICS System Programming Reference*. Be careful to specify GALENGTH or GAENTRYNAME on only the first **ENABLE** command, otherwise 'INVEXITREQ' may be returned.

Take into account the restrictions that apply to the use of CICS services, because these are dictated by the exit point itself rather than by the exit program. A command that can be issued from one exit point might cause problems when issued from a different exit point.

The global work area is associated with the exit program, rather than with the exit point: this means that the same global work area is used at each of the exit points at which the exit program is invoked.

Making applications threadsafe

When you make an application program threadsafe, you can use the open transaction environment, avoid TCB switching, and gain performance benefits.

Before you begin

To use threadsafe application programs, ensure that the system initialization parameter **FORCEQR** is not set to YES. **FORCEQR** forces programs defined as threadsafe to run on the QR TCB, and it might be set to YES as a temporary measure while problems connected with threadsafe-defined programs are investigated and resolved.

Also, select an appropriate setting for the system initialization parameter **FCQRONLY** in your file-owning CICS regions. If **FCQRONLY** is set to YES, CICS forces all file control requests in the CICS region to run on the QR TCB.

- If you use IPIC connections to function ship file control requests to remote CICS TS 4.2 or later regions, to improve performance for those connections set **FCQRONLY** to NO in the file-owning regions.
- If you use only MRO links or ISC over SNA connections, or your file-owning regions are earlier than CICS TS 4.2, set **FCQRONLY** to YES in the file-owning regions.

If you are using CICS intercommunication to make requests for functions or programs to run in remote CICS systems, choose IP interconnectivity (IPIC) over TCP/IP connections between the CICS systems to provide optimal support for threadsafe applications. With IPIC connections, CICS uses an open TCB to run the mirror program that manages the request on the remote CICS system, providing improved throughput. With other connection types, CICS does not use an open TCB to run the mirror program. The EXEC CICS LINK command, used for

distributed program link (DPL), is threadsafe for IPIC connections to remote CICS regions where a long-running mirror is used, but not for other connection types.

About this task

Threadsafe programs explains what it means for a program to be threadsafe, and the circumstances when TCB switching takes place between open TCBs and the QR TCB.

To make an application program threadsafe and enable it to remain on an open TCB, use the following procedure.

Procedure

1. Define the program to CICS as threadsafe, by specifying CONCURRENCY(THREADSAFE) in the PROGRAM resource definition.

   For a program that is defined as OPENAPI, CICS requires the CONCURRENCY(THREADSAFE) option. Only code that has been defined as threadsafe is permitted to run on open TCBs. By defining a program to CICS as threadsafe, you are specifying only that the application logic is threadsafe, not that all the EXEC CICS commands included in the program are threadsafe. CICS can ensure that EXEC CICS commands are processed safely by using TCB switching, but, to permit your program to run on an open TCB, CICS needs you to guarantee that your application logic is threadsafe.

   Alternatively, you can define the program as CONCURRENCY(REQUIRED) to enable your program to run from the start on an open TCB. Programs defined as CONCURRENCY(REQUIRED) must be coded to threadsafe standards as they must always run on an open TCB. The type of open TCB used depends on the API setting.
2. Ensure that the program logic, that is, the native language code between the EXEC CICS commands, is threadsafe.

   If you define a program to CICS as threadsafe but include application logic that is not threadsafe, the results are unpredictable, and CICS cannot protect your program from the possible consequences. To make your program logic threadsafe, you must use appropriate serialization techniques when accessing shared resources, to prohibit concurrent access to those resources. When you use EXEC CICS commands to access resources such as files, transient data queues, temporary storage queues, and DB2® tables, CICS ensures threadsafe processing, but for any resources that are accessed directly by user programs, such as shared storage, the user program must ensure threadsafe processing.

   Typical examples of shared storage are the CICS CWA, the global work areas for global user exits, and storage acquired explicitly by the application program with the shared option. Check whether your application programs use these types of shared storage by looking for occurrences of the following EXEC CICS commands:

   - ADDRESS CWA
   - EXTRACT EXIT
   - GETMAIN SHARED

   The load module scanner utility includes a sample table, DFHEIDTH, to help you identify CICS commands that give access to shared storage. Although some of these commands are themselves threadsafe, they all give access to global storage areas, so the application logic that follows these commands and uses the global storage areas can potentially not be threadsafe. To ensure that it is

threadsafe, an application program must include the necessary synchronization logic to guard against concurrent update.

Tip: When identifying programs that use shared resources, also include any program that modifies itself. Such a program is effectively sharing storage and you must consider it at risk.

You can use the following techniques to provide threadsafe processing when accessing a shared resource:

- Retry access, if the resource has been changed concurrently by another program, using the compare and swap instruction.
- Enqueue on the resource, to obtain exclusive control and ensure that no other program can access the resource, using one of the following techniques:
  - An EXEC CICS ENQ command, in an application program.
  - An XPI ENQUEUE function call to the CICS enqueue (NQ) domain, in a global user exit program.
  - An MVS service such as ENQ, in an open API task-related user exit only when L8 TCBs are enabled for use. Note that the use of MVS services in an application that can run under the QR TCB might result in performance degradation because of the TCB being placed in a wait.
- Perform accesses to shared resources only in a program that is defined as quasi-reentrant, by linking to the quasi-reentrant program using the EXEC CICS LINK command. This technique applies to threadsafe application programs and open API task-related user exits only. A linked-to program defined as quasi-reentrant runs under the QR TCB and can take advantage of the serialization provided by CICS quasi-reentrancy. Note that, even in quasi-reentrant mode, serialization is provided only for as long as the program retains control and does not wait.
- Place all transactions that access the shared resource into a restricted transaction class (TRANCLASS), one that is defined with the number of active tasks specified as MAXACTIVE(1). This approach effectively provides a very coarse locking mechanism, but might have a severe impact on performance.

Note: Although the term threadsafe is defined in the context of individual programs, a user application as a whole can be considered threadsafe only if all the application programs that access shared resources obey the rules. A program that is written correctly to threadsafe standards cannot safely update shared resources if another program that accesses the same resources does not obey the threadsafe rules.

3. For best performance, ensure that the program uses only threadsafe EXEC CICS commands.

   If you include an EXEC CICS command that is not threadsafe in a program that is defined as threadsafe and running on an open TCB, CICS switches back from the open TCB to the QR TCB to ensure that the command is processed safely. The results of your application are not affected, but its performance might be affected.

   The commands that are threadsafe are indicated in the command descriptions of the CICS API and SPI command topics with the statement “This command is threadsafe”. They are also listed in Threadsafe commands in Reference -> Application development and Threadsafe SPI commands in Reference > System programming.

   The load module scanner utility includes a sample table, DFHEIDNT, to help identify any CICS commands in your applications that are not threadsafe.

4. If any user exit programs are in the execution path used by the program, for best performance ensure that they are also coded to threadsafe standards and defined to CICS as threadsafe. These exits might be dynamic plan exits, global user exits, or task-related user exits. Also check that user exit programs supplied by any vendor software are coded to threadsafe standards and defined to CICS as threadsafe.

   A threadsafe user exit program can be used on the same open TCB as a threadsafe application that calls it, and it can use non-CICS APIs without having to create and manage subtask TCBs, and exploit the open transaction environment for itself. If any user exit programs in the execution path used by the program are not threadsafe, CICS switches to the QR TCB to run them, which might be detrimental to the application's performance.

   Be aware of the following important user exits:

   - The global user exits XEIIN and XEIOUT are called before and after EXEC CICS commands.
   - The global user exit XPCFTCH is called before a program defined to CICS receives control.
   - For CICS DB2 requests, the CICS DB2 task-related user exit DFHD2EX1 is threadsafe. Other important exits for CICS DB2 requests include the default dynamic plan exit DSNCUEXT, which is not defined as threadsafe, the alternative dynamic plan exit DFHD2PXT, which is defined as threadsafe, and the global user exits XRMIIN and XRMIOUT.
   - If you function ship CICS file control, transient data, or temporary storage requests to a remote CICS region over an IPIC connection, the requests are threadsafe and can run in the remote CICS region on an open TCB. Any global user exit programs that are called in the remote CICS region for file control, transient data, or temporary storage requests must be enabled as threadsafe programs for the best performance.

   a. To define a user exit program to CICS as threadsafe, you can specify appropriate attributes in its PROGRAM resource definition.
      - For a task-related user exit program, specify OPENAPI and THREADSAFE, or just THREADSAFE.
      - For a global user exit program, you cannot use OPENAPI, but you can specify THREADSAFE.

      If you specify CONCURRENCY(REQUIRED) on a global user exit program or task-related user exit program, CICS treats the program as if you had specified CONCURRENCY(THREADSAFE).

   b. Alternatively, to define a user exit program to CICS as threadsafe, you can specify appropriate options when you enable the program by using the **EXEC CICS ENABLE PROGRAM** command.
      - For a task-related user exit program, specify OPENAPI or THREADSAFE.
      - For a global user exit program, you cannot use OPENAPI, but you can specify THREADSAFE.

      When you enable an exit program using the OPENAPI or THREADSAFE option, this action indicates to CICS that the program logic is threadsafe, so CICS overrides the CONCURRENCY setting on the program definition for the exit and treats the exit program as threadsafe.

   c. To define a first-phase PLT global user exit program as threadsafe, specify THREADSAFE on the **EXEC CICS ENABLE PROGRAM** command. To ensure that global user exit programs (such as those that run at the recovery exit points) are available as early as possible during CICS initialization, it is common practice to enable them from first-phase PLT programs. Because first-phase

PLT programs run so early in CICS initialization, you cannot use installed PROGRAM resource definitions or the program autoinstall user program to define the exit programs. CICS automatically installs exit programs that are enabled from first-phase PLT programs with CONCURRENCY(QUASIRENT). However, the setting on the **EXEC CICS ENABLE PROGRAM** command overrides the CONCURRENCY(QUASIRENT) setting on the system-autoinstalled program definition.

Using the task token UEPTSTOK

UEPTSTOK is an exit-specific parameter that is passed on a number of user exit points. It provides the address of a 4-byte area that you can use to pass information between successive interval control requests in the same task. For example, if you need to pass information between successive invocations of the XFCREQ exit, you can use UEPTSTOK to do so.

UEPTSTOK is set to address the EISEXITT field (the task lifetime token in the EIS). The expectation is that this field is set to address an area of storage that is used by the exits for the task. As such, this task lifetime storage can potentially be shared between different exit programs for the life of a task.

Unless exit programs cooperate in their usage of UEPTSTOK, unpredictable results can occur when it is used to address exit-specific data. An approach for sharing this token by multiple exit programs is as follows:

- The UEPTSTOK token is the first of a chain of addresses, the subsequent editions of which are at the front of each piece of storage chained. Exit programs acquire storage and chain the addresses of this storage from UEPTSTOK.
- Bytes 0-3 of each piece of storage is a pointer to the next piece of storage chained. Bytes 4-11 contain the name of the exit program which acquired the storage (or some other constant identifying whose storage it is).

Implement a mechanism such as this for exit programs that use UEPTSTOK, to ensure that the field can be shared between multiple exit programs that are invoked by the same task.

Note: Typically, exit programs detect that there is not a currently chained piece of storage for their specific use when driven for the first time by a task, and this makes them acquire and add a piece of storage to the chain addressed by UEPTSTOK. When they add a piece of storage to the chain, it can be done to the front or end of the chain; what matters is that the chain is updated to reflect the newly added storage address and exit identifier.

Sample global user exit programs

CICS provides two sets of sample and example global user exit programs.

1. A set that shows you how to do basic things, such as:
   - Enable a global user exit program and allocate a global work area
   - Use EXEC CICS and XPI commands in a global user exit program.
2. A set of samples for use at specific global user exit points.

The source of all the sample programs and any associated copy books is supplied in the CICSTS52.CICS.SDFHSAMP library. You can use the supplied programs as models on which to base your own versions.

Basic sample and example programs

CICS supplies a number of samples to use when creating your global user exit programs.

Global work area (GWA) sample exit programs

The global work area sample exit programs and copy books provide examples of how to work with global work areas and global user exits. Three sample exit programs are provided.

This set of sample programs shows you how to:

- Enable a global user exit program and allocate a global work area (GWA).
- Obtain the address of an exit program's GWA.
- Access CICS system information, and make that information available to other global user exit programs.
- Share a GWA between global user exit programs, thereby making the information it contains available to more than one program, and overcoming limitations on the size of GWAs.
- Access information held in a global user exit program's GWA.

The GWA sample programs and copy books are:

DFH$PCEX
: A sample global user exit program, designed to be invoked at the XPCFTCH exit.

  CICS also provides copy book DFH$PCGA for use in this sample program.

DFH$ZCAT
: A sample global user exit program, designed to be invoked at the XZCATT exit.

  CICS also provides copy book DFH$ZCGA for use in this sample program.

DFH$PCPI and DFH$PCPL
: DFH$PCPI is designed to be invoked during program list table post initialization (PLTPI) processing, and is described in "Sample program for global user exits (DFH$PCPI)."

  DFH$PCPL is a dummy program, invoked by DFH$PCPI, that causes the XPCFTCH user exit to be driven.

Sample program for global user exits (DFH$PCPI)

The DFH$PCPI sample program provides an example of how to work with global work areas.

DFH$PCPI consists of three main sections:

- Section 1 obtains and processes any parameters passed to DFH$PCPI on the **INITPARMS** system initialization parameter.
- Section 2 shows how to use standard CICS facilities to obtain system information, and make that information available to a global user exit program. Section 2 performs the following processing:
  - Uses the **EXEC CICS ENABLE PROGRAM** command to enable the XPCFTCH sample user exit program, DFH$PCEX, and allocate it a global work area.
  - Uses the **EXEC CICS EXTRACT EXIT** command to obtain the address of the DFH$PCEX global work area.

  - Obtains CICS system information, and places it in the DFH$PCEX global work area. The information obtained includes:
    - Job name
    - APPLID
    - Sysid
    - CICS release
    - Date in various formats, including DATFORM
    - Address of the common work area (CWA)
    - CICS startup type (COLD, WARM)

    Most of the preceding information is obtained using EXEC CICS API commands such as:
    - **INQUIRE SYSTEM**
    - **ASSIGN**
    - **ADDRESS**
    - **ASKTIME**
    - **FORMATTIME**
  - Uses the START option of the **EXEC CICS ENABLE PROGRAM** command to make DFH$PCEX available for execution. This causes DFH$PCEX to be driven for all **LINK** and **XCTL** commands.
  - Links to the dummy program, DFH$PCPL, in order to drive DFH$PCEX.
  - Uses the STOP option of the **EXEC CICS DISABLE PROGRAM** command to make DFH$PCEX unavailable for execution. Note that this leaves the DFH$PCEX global work area still allocated and accessible through the **EXEC CICS EXTRACT EXIT** command.
- Section 3 of DFH$PCPI shows how to share the system information in an exit program's global work area with other exit programs. In doing so it demonstrates how application programs can access the same information by means of the **EXEC CICS EXTRACT EXIT** command.

  This section also shows how to use CICS shared storage to overcome the limitation of 32 KB on the size of a global work area. The program uses GETMAIN to obtain an area of 64 KB in 24-bit storage (below 16 MB) and an area of 128 KB in 31-bit storage (above 16 MB). The use of shared storage enables the second user exit program (DFH$ZCAT) to use a work area of only 12 bytes in 31-bit storage.

  The section performs the following processing:
  - Uses **EXEC CICS ENABLE** to enable the DFH$ZCAT user exit program, and allocate it a global work area.
  - Uses **EXEC CICS EXTRACT EXIT** to obtain the address of the DFH$ZCAT global work area.
  - Stores the address of the DFH$PCEX global work area in the DFH$ZCAT global work area.
  - Uses GETMAIN to obtain the shared 24-bit and 31-bit storage, and stores the addresses in the DFH$ZCAT global work area.

Sample program definitions

RDO provides the traditional CICS way of defining resources to CICS systems. In this case RDO defines the sample programs to the CSD.

Here are some examples of RDO sample program definitions:

```
DEFINE PROGRAM(DFH$PCEX) GROUP(EXITGRP)
        LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
        USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
        EXECKEY(CICS)

DEFINE PROGRAM(DFH$PCPI) GROUP(EXITGRP)
        LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
        USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
        EXECKEY(CICS)

DEFINE PROGRAM(DFH$PCPL) GROUP(EXITGRP)
        LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
        USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
        EXECKEY(CICS)

DEFINE PROGRAM(DFH$ZCAT) GROUP(EXITGRP)
        LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
        USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
        EXECKEY(CICS)
```

DFH$PCPI is designed to be run as a PLT program. If you write a similar program, you should define it in the **second** part of the PLTPI list (after the PROGRAM=DFHDELIM entry). Information about how to do this is in the *CICS Resource Definition Guide*.

Sample program (DFH$XTSE)

This is additional sample program that demonstrates how to work with EXEC CICS commands and user exits.

A

- Use **EXEC CICS** commands in a global user exit program
- Use **EXEC CICS** commands and XPI calls in the same exit program
- Modify the command parameter list in EXEC interface exits such as XTSEREQ and XICEREQ.

For more information, see Appendix G, “The example program for the XTSEREQ global user exit, DFH$XTSE,” on page 917.

Sample programs for specific exits

Certain user exit points have sample programs that demonstrate how you can use the exit point in your global user exit programs.

Application associated data sample exit program (DFH$APAD)

CICS includes a sample global user exit program for the application associated data exit. The sample program is DFH$APAD.

DFH$APAD
: This sample global user exit program is invoked at the XAPADMGR exit. The program shows how you can add user information to a transaction's Associated Data Origin Descriptor or retrieve existing information found in the association data, for purposes such as auditing or accounting of workloads.

Related reference:

“Application association data exit in the AP domain (XAPADMGR)” on page 40
Use the XAPADMGR exit for distributed transactions. XAPADMGR allows you to add user information to the association data of a task, at the point of origin of the distributed transaction. This information could be used later, for example, as a search key for processing carried out through CICSPlex SM.

Basic mapping support sample exit program (DFH$BMXT)

CICS includes a sample global user exit program for the basic mapping support user exits. The sample program is DFH$BMXT.

DFH$BMXT
: This sample global user exit program is invoked at the XBMIN and XBMOUT exits, and shows how you can use the exits to modify mapped input and output data.

Related concepts:

"Basic mapping support exits (XBMIN, XBMOUT)" on page 41
Two basic mapping support exits are provided: XBMIN and XBMOUT. The XBMIN exit allows you to intercept a RECEIVE MAP request after BMS has successfully processed the request. The XBMOUT exit allows you to intercept a SEND MAP request after BMS has successfully processed the request; or, if cumulative mapping is in progress, on completion of each page of output.

Data tables sample exit programs (DFH$DTAD, DFH$DTLC, DFH$DTRD)

CICS includes three sample global user exit programs, one for each of the data tables exit points. The sample programs are DFH$DTAD, DFH$DTLC, DFH$DTRD.

These sample programs are:

DFH$DTAD
: A sample global user exit program, designed to be invoked at the XDTAD exit.

DFH$DTLC
: A sample global user exit program, designed to be invoked at the XDTLC exit.

DFH$DTRD
: A sample global user exit program, designed to be invoked at the XDTRD exit.

DFH$DTAD, DFH$DTLC, and DFH$DTRD are listed in the CICS Shared Data Tables Guide.

Related concepts:

"Data tables management exits XDTRD, XDTAD, and XDTLC" on page 48
Data tables management exits apply to both CICS shared data tables and CICS coupling facility data tables.

Dump domain sample exit program (DFH$XDRQ)

CICS includes a dump domain sample global user exit program. The sample prom is DFH$XDRQ.

DFH$XDRQ
: A sample global user exit program, designed to be invoked at the XDUREQ exit. The sample shows you how to manipulate the dump table entry, and how to permit or suppress the dump.

Related concepts:

"Dump domain exits XDUREQ, XDUREQC, XDUCLSE, and XDUOUT" on page 68
There are four exits in the dump domain that you can use to capture information before and after a transaction dump or system dump.

Related reference:

"Exit XDUREQ" on page 68
Exit XDUREQ is invoked immediately before a system or transaction dump is taken.

Enqueue EXEC interface sample exit program (DFH$XNQE)

CICS includes a sample global user exit program for the enqueue EXEC interface. The sample program is DFH$XNQE.

DFH$XNQE
: A sample global user exit program, designed to be invoked at the XNQEREQ and XNQEREQC exits. The program demonstrates three ways of adding a SCOPE value to EXEC CICS ENQ and DEQ requests, to make the requests apply to multiple regions within the sysplex.

Related concepts:

"Sample exit program, DFH$XNQE" on page 82
CICS supplies a sample exit program, DFH$XNQE, for the XNQEREQ exit.

File control state sample exit program (DFH$REQC)

CICS includes a sample global user exit program for use with the file control state program. The sample program is DFH$REQC.

DFH$REQC
: A sample user exit program, designed to be invoked at the XFCSREQC exit. It allows you to check VSAM error flags after file requests have been acted upon and, optionally, to write messages and data areas to a system console or a transient data queue. See "The DFH$REQC sample global user exit program" on page 142.

Related concepts:

"The DFH$REQC sample global user exit program" on page 142
DFH$REQC provides sample processing for the file control state program global user exit, XFCSREQC. The exit program, if enabled at the XFCSREQC exit point, is invoked after a file ENABLE, DISABLE, OPEN, CLOSE, or CANCEL CLOSE request has been acted on.

File control recovery sample exit programs (DFH$FC*)

CICS includes three sample file control global user exit programs. The sample programs are DFH$FCBF, DFH$FCBV, DFH$FCLD.

DFH$FCBF
: A sample exit program designed to be invoked at the XFCBAIL exit for handling backout failures. See "DFH$FCBF sample global user exit program" on page 154.

DFH$FCBV
: A sample exit program designed to be invoked at the XFCBOVER exit; it allows you to decide whether to allow an update to be backed out, following a batch update run that has overridden retained locks. See "DFH$FCBV sample global user exit program" on page 158.

DFH$FCLD
: A sample exit program designed to be invoked at the XFCLDEL exit, which allows you to perform logical deletion of records from a VSAM ESDS data set or a BDAM data set, during backout. See "DFH$FCLD sample global user exit program" on page 160.

To define these programs include the supplied resource group DFH$FCB in your startup grouplist, or use CEDA to install DFH$FCB.

Related concepts:

"DFH$FCBF sample global user exit program" on page 154
DFH$FCBF provides sample processing for the file control backout failure global user exit, XFCBFAIL. The exit program, if enabled at the XFCBFAIL exit point, is invoked if an error occurs during backout of a file control update.

"DFH$FCBV sample global user exit program" on page 158
DFH$FCBV provides sample processing for the file control backout override global user exit, XFCBOVER.

"DFH$FCLD sample global user exit program" on page 160
DFH$FCLD provides sample processing for the file control logical delete global user exit, XFCLDEL.

Function-shipping/DPL queue control sample exit program (DFHXIS)

CICS includes a sample global user exit program that controls the queueing of function-shipping and DPL requests. The sample program is DFHXIS.

DFHXIS
: A sample global user exit program, designed to be invoked at the XISCONA exit.

Related concepts:

"The sample XISCONA global user exit program, DFHXIS" on page 173
A CICS-supplied sample exit program, DFHXIS,shows one way of limiting the queue of ALLOCATE requests, based on the information passed to the program.

HTTP client sample exit programs (DFH$WB*)

CICS includes four sample programs for use with the Web domain exits XWBOPEN and XWBAUTH. The sample programs are DFH$WBPI, DFH$WBEX, DFH$WBX1, DFH$WBX2.

The XWBOPEN exit is invoked during processing of **EXEC CICS WEB OPEN** and **EXEC CICS INVOKE SERVICE** commands. XWBAUTH is called during processing of an **EXEC CICS WEB SEND** and **EXEC CICS WEB CONVERSE** commands. Both exits are used in making HTTP client requests from CICS as an HTTP client, which is a facility provided by CICS Web support.

The following sample exit programs are shipped in the CICS sample library, SDFHSAMP:

- DFH$WBPI, described in "DFH$WBPI"
- DFH$WBEX, described in "DFH$WBEX" on page 26
- DFH$WBX1, described in "DFH$WBX1" on page 26
- DFH$WBX2, described in "DFH$WBX2" on page 27
- DFH$WBGA, a copybook to map the global work area used by the DFH$WBPI, DFH$WBX1, DFH$WBX2, and DFH$WBEX samples.

DFH$WBPI

This program, whose purpose is to initialize the supplied Web-related global user exits, is specified in the PLTPI and is invoked during the CICS post-initialization phase. It is specified with the **INITPARM** system initialization parameter as follows:

```
INITPARM=(DFH$WBPI='PROXY=proxyurl,LDAPBIND=profilename,STS=sts-server-url')
```

where

PROXY=proxyurl
This optional keyword stores the URL (in the form http://proxyserver) of a proxy server into the Web global work area, then enables the supplied DFH$WBEX sample program as the XWBOPEN global user exit.

LDAPBIND=profilename
This optional keyword stores the name of an LDAP bind profile into the Web global work area, then enables the supplied DFH$WBX1 sample program as the XWBAUTH global user exit.

Note that you cannot specify both LDAPBIND and STS. To do so causes DFH$WBPI to abend with code WBPI. Message DFHSI1580D is issued, which may cause CICS to be terminated.

STS=sts-server-url
This optional keyword stores the URL (usually in the form https://sts-server) of a Secure Token Service into the Web global work area, then enables the supplied DFH$WBX2 sample program as the XWBAUTH global user exit.

Note that you cannot specify both STS and LDAPBIND. To do so causes DFH$WBPI to abend with code WBPI. Message DFHSI1580D is issued, which may cause CICS to be terminated.

Note that the total length of the INITPARM quoted text cannot exceed 60 characters.

DFH$WBEX

This sample global user exit program is designed to check the host name specified on the **EXEC CICS WEB OPEN** command, and make any host name starting with `www` use a proxy server if a proxy server name is specified in the global work area.

If all the requests from your CICS system should use a single proxy server, you can use the proxy server name from the **INITPARM** system initialization parameter, that DFH$WBPI used to initialize the global work area.

- The proxy name must be specified as:

```
INITPARM=(DFH$WBPI='PROXY=proxyurl')
```

where proxyurl is the URL if a proxy server. If you use a number of proxy servers or want to apply a security policy to different host names, you can load or build a table that matches host names to appropriate proxy servers or marks them as barred, which can then be used as a look-up table during processing of the **EXEC CICS** WEB OPEN command.

DFH$WBX1

This sample global user exit program has the following functions:

- If a GLUE global workarea is provided and it contains a non-zero LDAP connection token, it uses that token in subsequent SEARCH requests.
- If the exit is called at the XSTERM (system termination) exit point, it terminates the LDAP connection by invoking the DFHDDAP UNBIND_LDAP function. Otherwise, it obtains a connection token by issuing DFHDDAP BIND_LDAP and stores it in the global workarea. The LDAPBIND profile specified in the INITPARM parameter for DFH$WBPI is used to obtain LDAP credentials.
- Composes a distinguished name in the following format: `racfcid=uuuuuuuu, ibm-httprealm=rrrrrrrr, labeledURI=xxxxxxxx, cn=BasicAuth` where:

racfcid=uuuuuuuu
is the current userid, obtained from UEPUSER

ibm-httprealm=rrrrrrrr
is the HTTP 401 realm, obtained from UEPREALM (if this exists)

labeledURI=xxxxxxxx
is the target URL, obtained by concatenating "http://" with the hostname from UEPHOST and the path from UEPPATH

cn=BasicAuth
is an arbitrary suffix that is configured into the LDAP server for the purpose of storing Basic Authentication credentials.

- Issues DFHDDAP SEARCH_LDAP with this distinguished name
- If the SEARCH_LDAP fails, DFH$WBX1 removes the REALM parameter from the distinguished name and repeats the search. If the search fails again, DFH$WBX1 removes the **UID** parameter from the distinguished name and repeats the search. If the search fails for the third time, DFH$WBX1 returns from the exit with return code UERCERR.
- If the search was successful, issue DFHDDAP START_BROWSE_RESULTS
- Obtains the target **username** credential by obtaining the value of the **UID** attribute with DFHDDAP GET_ATTRIBUTE_VALUE. This is set into the response area provided by UEPUSNM.
- Obtains the target **password** credential by obtaining the value of the **UserPassword** attribute with DFHDDAP GET_ATTRIBUTE_VALUE. This is set into the response area provided by UEPPSWD.
- Releases the browse storage by issuing DFHDDAP END_BROWSE_RESULTS
- If the bind token was not stored in the global workarea, terminate the LDAP session by issuing DFHDDAP UNBIND_LDAP
- If all is successful, DFHWBX1 returns from the exit with return code UERCNORM.

DFH$WBX2

This sample global user exit program has the following functions:

- Obtains the destination HTTP host from UEPHOST/UEPHOSTL and the destination HTTP path from UEPPATH/UEPPATHL, and uses them to construct the URL of the HTTP server for which the basic authentication credentials are required, as follows: `http://hostname/pathname`.
- If a realm exists (that is, if UEPREALML is non-zero), DFH$WBX2 appends the realm from UEPREALM to the URL created above, separated by a number sign (#) to make it look like a URL fragment identifier, as follows: `http://hostname/pathname#realm`. If necessary, the realm is URL-encoded.
- Stores the URL in the DFHWS-SERVICEURI container in the DFHWSTC-V1 channel.
- Stores the URL of the Security Token Service (STS), obtained from the global work area, in the DFHWS-STSURI container in the DFHWSTC-V1 channel.
- Stores architecturally appropriate URIs into the DFHWS-STSACTION and DFHWS-TOKENTYPE containers in the DFHWSTC-V1 channel.
- Constructs a username token from the caller's userid passed in UEPUSER, and store it in the DFHWS-IDTOKEN container in the DFHWSTC-V1 channel.
-

Note: It is not necessary to specify a pipeline in the DFHWS-PIPELINE container. The pipeline is dynamically created by DFHPIRT when CHANNEL(DFHWSTC-V1) is specified on the command.

- Links to DFHPIRT, specifying CHANNEL(DFHWSTC-V1), after constructing all the required containers. This sends the request to the STS and receives the response into the DFHWS-RESTOKEN container.
- If the LINK was successful, DFHWBX2 recovers the response from the DFHWS-RESTOKEN container, which contains an username token in XML format.
- Extracts the username and password from this token, and returns them as responses from the user exit in UEPUSNM/UEPUSNML and UEPPSWD/UEPPSWDL. Returns from the user exit with return code UERCNORM.
- If the LINK was unsuccessful, or if a SOAP fault response is returned, DFH$WBX2 returns from the exit with return code UERCERR.

The above implementation assumes that the STS server is configured to respond with an appropriate username token when presented with the URI formatted by DFH$WBX2 in the `AppliesTo` element of the STS issue request.

Related concepts:

"HTTP client open and send exits: XWBAUTH, XWBOPEN and XWBSNDO" on page 164
Exits XWBAUTH, XWBOPEN and XWBSNDO are invoked during processing of **EXEC CICS WEB CONVERSE**, **EXEC CICS WEB OPEN**, **EXEC CICS INVOKE SERVICE**, and **EXEC CICS WEB SEND** commands. They are used in making HTTP client requests from CICS as an HTTP client, which is a facility provided by CICS Web support.

Interval control EXEC interface sample exit program (DFH$ICCN)

CICS includes an interval control EXEC interface sample exit program. The sample program is DFH$ICCN.

DFH$ICCN
: A sample global user exit program, designed to be invoked at the XICEREQ exit. DFH$ICCN is for use in a distributed routing environment, where you want to cancel a previously-issued interval control request but have no way of knowing to which region to direct the CANCEL. For examples of situations which DFH$ICCN is designed to cope with, see Canceling interval control requests, in the *CICS Intercommunication Guide*.

Related concepts:

"Example of how XICEREQ and XICEREQC can be used" on page 199
In this example, XICEREQ and XICEREQC are used to route START requests to a number of different CICS regions to provide a simple load balancing mechanism. The example shows only the capabilities of the exits; it is not intended to indicate an ideal way of achieving the function.

Log manager domain sample exit program (DFH$LGLS)

CICS includes a sample global user exit program for the XLGSTRM exit point. The sample program is DFH$LGLS.

DFH$LGLS
: The sample program shows you how to change parameters such as the model log stream name used by the MVS system logger when it creates a new log stream dynamically. The sample is described in "The sample program for the XLGSTRM exit, DFH$LGLS" on page 206.

Related concepts:

"The sample program for the XLGSTRM exit, DFH$LGLS" on page 206
The sample program, DFH$LGLS, shows you how to access and change some of the parameters passed to an XLGSTRM exit program.

Message domain sample exit programs (DFH$SXP*)

CICS includes six sample programs that are called at the XMEOUT exit to complete specified tasks such as rerouting a console message to a transient data queue. The sample programs are DFH$SXP1, DFH$SXP2, DFH$SXP3, DFH$SXP4, DFH$SXP5, DFH$SXP6.

DFH$SXP*

A set of sample global user exit programs designed to be invoked at the XMEOUT exit (where * is 1 through 6).

Related concepts:

"The sample XMEOUT global user exit programs" on page 210
CICS includes six sample programs that show how to use the XMEOUT exit to suppress or reroute messages.

MRO and APPC session queue management sample exit program (DFH$XZIQ)

CICS includes a sample exit program that implements the basic functions provided by the QUEUELIMIT and MAXQTIME parameters on a connection resource definition. The sample program is DFH$XZIQ.

The parameters are passed to the XZIQUE global user program, which can change the way in which these parameters are used:

DFH$XZIQ

A sample global user exit program, designed to be invoked at the XZIQUE exit, is described in "XZIQUE exit for managing MRO and APPC intersystem queues" on page 318.

See "Sample exit program design" on page 326 for more details.

Related concepts:

"Designing an XZIQUE global user exit program" on page 325

Session queue management sample exit program for IPIC connections (DFH$XISQ)

CICS includes a sample program that implements the basic function provided by the QUEUELIMIT and MAXQTIME parameters on a connection resource definition. The sample program is DFH$XISQ.

The parameters are passed to the XISQUE global user program, which can change the way in which these parameters are used.

DFH$XISQ

A sample global user exit program, designed to be invoked at the XISQUE exit, which is described in"XISQUE exit for managing IPIC intersystem queues" on page 327.

See "The sample exit program, DFH$XISQ" on page 332 for more details.

Related concepts:

"XISQUE exit for managing IPIC intersystem queues" on page 327
You can use the XISQUE exit to control queuing on IP interconnectivity (IPIC) connections.

IPIC queue control sample exit program (DFH£XISL)

CICS includes a sample exit program for controlling the queueing of requests destined for an IPIC connection. The sample program is DFH£XISL.

You use the XISQLCL sample global user exit program DFH£XISL to control the queueing of START NOCHECK requests that are scheduled for an IPIC connection.

DFH£XISL
: A sample global user exit program, designed to be invoked at the XISQLCL exit, which is described in "The XISQLCL exit" on page 176.

See "Sample exit program design" on page 326 for more details.

Pipeline sample exit program (DFH$PIEX)

CICS includes a sample global user exit program that provides sample processing for the XWSPRROO global user exit. The sample program is DFH$PIEX.

DFH$PIEX
: If you enable this sample program, it is invoked in a Web service provider pipeline after the Web service provider application has finished processing and returned to CICS, but before CICS creates the response message SOAP body. The sample program retrieves the name of the Web service, the name of the transaction, and the Web service attributes.

Related reference:

"Exit XWSPRROO" on page 217
Use the XWSPRROO exit to access containers on the current channel after the Web services provider application issues the Web service response message and before CICS creates the body of the response message.

Transaction abend sample exit program (DFH$PCTA)

CICS includes a sample global user exit program for the XPCTA exit point. The sample program is DFH$PCTA.

DFH$PCTA
: This sample global user exit program is designed to be invoked at the XPCTA exit, to test whether the abend was caused by a storage protection exception condition. It is described in "The sample XPCTA global user exit program, DFH$PCTA" on page 248.

Related concepts:

"The sample XPCTA global user exit program, DFH$PCTA" on page 248
The sample program tests whether the abend was caused by the application program trying to overwrite CICS-key storage in the CDSA, ECDSA, ETDSA, or GCDSA while running in user key. If this was the case, the sample changes the execution key to CICS, and retries the failing instruction.

Terminal-not-known sample exit program (DFHXTENF)

CICS includes a sample global user exit program that handles terminal-not-known conditions arising from START and ATI requests. The sample program is DFHXTENF.

DFHXTENF
: A sample global user exit program, designed to be invoked at the XALTENF or XICTENF exit. The sample source code is shown in "The sample program for the XALTENF and XICTENF exits, DFHXTENF" on page 296.

Related concepts:

"The sample program for the XALTENF and XICTENF exits, DFHXTENF" on page 296
DFHXTENF is a sample program that can be used for the XALTENF and XICTENF exits.

Global user exit points (in alphabetical order)

For each exit, this table shows the exit name, the module or domain, where or when the exit is invoked, and includes a link to additional information.

Table 3. Alphabetical list of global user exit points

Exit name	Module or domain	Where or when invoked	Topic
XAKUSER	Activity keypoint program	Immediately before the 'end of keypoint' record is written.	"Activity keypoint program exit (XAKUSER)" on page 39
XALCAID	Terminal allocation program	Whenever an AID with data is canceled.	"Terminal allocation program exit XALCAID" on page 286
XALTENF	Terminal allocation program	When an ATI request from transient data or interval control requires a terminal that is unknown in this system.	"Exit XALTENF" on page 292
XAPADMGR	Application domain	When a nonsystem task that has no inherited Associated Data Origin Descriptor data is attached.	"Application association data exit in the AP domain (XAPADMGR)" on page 40
XBMIN	Basic Mapping Support	When an input mapping operation completes successfully.	"Exit XBMIN" on page 43
XBMOUT	Basic Mapping Support	When a page of output has been built successfully.	"Exit XBMOUT" on page 43
XDLIPOST	DL/I interface program	On exit from the DL/I interface program.	"Exit XDLIPOST" on page 61
XDLIPRE	DL/I interface program	On entry to the DL/I interface program.	"Exit XDLIPRE" on page 59
XDSAWT	Dispatcher domain	After an operating system wait.	"Exit XDSAWT" on page 57
XDSBWT	Dispatcher domain	Before an operating system wait.	"Exit XDSBWT" on page 57
XDTAD	Data tables management	When a write request is issued to a data table.	"Exit XDTAD" on page 51
XDTLC	Data tables management	At the completion of loading of a data table.	"Exit XDTLC" on page 53
XDTRD	Data tables management	During the loading of a data table, whenever a record is retrieved from the source data set.	"Exit XDTRD" on page 49
XDUCLSE	Dump domain	After the domain closes a transaction dump data set.	"Exit XDUCLSE" on page 74
XDUOUT	Dump domain	Before the domain writes a record to the transaction dump data set.	"Exit XDUOUT" on page 74

Table 3. Alphabetical list of global user exit points (continued)

Exit name	Module or domain	Where or when invoked	Topic
XDUREQ	Dump domain	Before the domain takes a system or transaction dump.	"Exit XDUREQ" on page 68
XDUREQC	Dump domain	After a system or transaction dump has been taken (or failed or been suppressed).	"Exit XDUREQC" on page 71
XEIIN	EXEC interface program	Before any EXEC CICS API or SPI command runs.	"Exit XEIIN" on page 86
XEIOUT	EXEC interface program	After any EXEC CICS API or SPI command runs.	"Exit XEIOUT" on page 87
XEISPIN	EXEC interface program	Before any EXEC CICS SPI command *except* EXEC CICS ENABLE, EXEC CICS DISABLE, EXEC CICS EXTRACT EXIT, EXEC CICS PERFORM DUMP, or EXEC CICS RESYNC ENTRYNAM runs.	"Exit XEISPIN" on page 86
XEISPOUT	EXEC interface program	After any EXEC CICS SPI command *except* **EXEC CICS ENABLE**, **EXEC CICS DISABLE**, **EXEC CICS EXTRACT EXIT**, **EXEC CICS PERFORM DUMP**, or **EXEC CICS RESYNC ENTRYNAME** runs.	"Exit XEISPOUT" on page 88
XEPCAP	Event capture	Before an event is captured by CICS event processing.	"Event capture exit XEPCAP" on page 83
XFAINTU	3270 bridge facility management program	When a bridge facility is created or deleted.	"Bridge facility exit XFAINTU" on page 46
XFCAREQ	File control EXEC interface program	Before CICS processes a file control SPI request.	"File control EXEC interface SPI exits XFCAREQ and XFCAREQC" on page 118
XFCAREQC	File control EXEC interface program	After a file control SPI request has completed.	"File control EXEC interface SPI exits XFCAREQ and XFCAREQC" on page 118
XFCBFAIL	File control recovery control program	When an error occurs during the backout of a UOW.	"Exit XFCBFAIL, file control backout failure exit" on page 150
XFCBOUT	File control recovery control program	When CICS is about to back out a file update.	"Exit XFCBOUT, file control backout exit" on page 155
XFCBOVER	File control recovery control program	When CICS is about to skip backout of a UOW because a batch program has overridden RLS retained lock protection and opened a data set for batch processing.	"Exit XFCBOVER, file control backout override exit" on page 156
XFCFRIN	File control domain	Before a file control request runs.	"Exit XFCFRIN" on page 94
XFCFROUT	File control domain	After a file control request runs.	"Exit XFCFROUT" on page 100

Table 3. Alphabetical list of global user exit points (continued)

Exit name	Module or domain	Where or when invoked	Topic
XFCLDEL	File control recovery control program	When backing out writes to a VSAM ESDS or a BDAM data set.	"Exit XFCLDEL, file control logical delete exit" on page 159
XFCNREC	File control open/close program	When a mismatch is detected between the backout recovery setting for a file and its associated data set during file open processing.	"File control open/close program exit XFCNREC" on page 143
XFCQUIS	File control quiesce send program	On completion, successful or failed, of a SET DSNAME QUIESCESTATE command.	"File control quiesce send exit XFCQUIS" on page 147
XFCREQ	File control EXEC interface program	Before CICS processes a file control API request.	"Exit XFCREQ" on page 116
XFCREQC	File control EXEC interface program	After a file control API request has completed.	"Exit XFCREQC" on page 117
XFCRLSCO	File control RLS coexistence program	When the opening of a VSAM RLS file or non-RLS read-only file otherwise fails with an RLS coexistence failure.	"File control RLS coexistence program exit XFCRLSCO" on page 161
XFCSREQ	File control file state program	Before a file OPEN, CLOSE, ENABLE, or DISABLE command is attempted.	"File control file state program exits XFCSREQ and XFCSREQC" on page 133
XFCSREQC	File control file state program	After a file OPEN, CLOSE, CANCEL CLOSE, ENABLE, or DISABLE command has been completed.	"File control file state program exits XFCSREQ and XFCSREQC" on page 133
XFCVSDS	File control quiesce receive program	After RLS has informed CICS that processing is required as a result of a data set-related action occurring in the sysplex.	"File control quiesce receive exit, XFCVSDS" on page 145
XGMTEXT	"Good morning" message program	Before the "good morning" message is sent.	"Good morning message program exit (XGMTEXT)" on page 163
XICEREQ	Interval control EXEC interface program	Before CICS processes an interval control API request.	"Exit XICEREQ" on page 182
XICEREQC	Interval control EXEC interface program	After an interval control API request has completed.	"Exit XICEREQC" on page 185
XICERES	Interval control EXEC interface program	Before CICS processes a non-terminal-related EXEC CICS START request that has been dynamically routed to this region, where the routing region supports the "resource unavailable" (RESUNAVAIL) condition.	"Exit XICERES" on page 184
XICEXP	Interval control program	After expiry of an interval control time interval.	"Exit XICEXP" on page 180

Table 3. Alphabetical list of global user exit points (continued)

Exit name	Module or domain	Where or when invoked	Topic
XICREQ	Interval control program	At the start of the interval control program, before request analysis.	"Exit XICREQ" on page 178
XICTENF	Interval control program	When an EXEC CICS START command requires a terminal that is unknown in this system.	"Exit XICTENF" on page 294
XISCONA	Intersystem communication program	When a function shipping or DPL request is about to be queued because no sessions to the remote region are immediately available.	"Intersystem communication program exits, XISCONA, XISLCLQ, and XISQLCL" on page 171
XISLCLQ	Intersystem communication program	After an attempt to allocate a session for a function shipped START NOCHECK request fails because the remote system is not in service, a connection to the remote system cannot be established, or no sessions are immediately available and your XISCONA exit program has specified that the request is not to be queued in the issuing region.	"The XISLCLQ exit" on page 174
XISQLCL	Intersystem communication program	After an attempt to allocate a session for a START NOCHECK request, that is scheduled for an IPIC connection, fails because the IPIC connection is out of service, the IPIC connection is not acquired, or a session is not available and CICS does not queue the request for a new session.	"The XISQLCL exit" on page 176
XISQUE	To control the number of queued requests for sessions on IPCONNs	When: 1. An allocate request for a session on an IPCONN is about to be queued 2. An IP allocate request succeeds following previous suppression of queuing	"XISQUE exit for managing IPIC intersystem queues" on page 327
XLDELETE	Loader domain	After an instance of a program is released by CICS and just before the program is freed from storage.	"Exit XLDELETE" on page 201
XLDLOAD	Loader domain	After an instance of a program is brought into storage, and before the program is made available for use.	"Exit XLDLOAD" on page 200
XLGSTRM	Log manager domain	After the CICS log manager detects that a log stream does not exist, and before calling the MVS system logger to define the log stream.	"Log manager domain exit XLGSTRM" on page 202
XMEOUT	Message domain	Before a message is sent from the message domain to its destination.	"Exit XMEOUT" on page 208
XMNOUT	Monitoring domain	Before a record is either written to SMF or buffered before a write to SMF.	"Exit XMNOUT" on page 211
XNQEREQ	Enqueue EXEC interface program	Before CICS processes an enqueue API request.	"Exit XNQEREQ" on page 76
XNQEREQC	Enqueue EXEC interface program	After an enqueue API request has completed.	"Exit XNQEREQC" on page 77
XPCABND	Program control program	After a transaction abend and before a dump call is made.	"Exit XPCABND" on page 248

Table 3. Alphabetical list of global user exit points (continued)

Exit name	Module or domain	Where or when invoked	Topic
XPCERES	Program control program	Before CICS processes a program link or Link3270 bridge request that has been dynamically routed to this region, where the routing region supports the "resource unavailable" (RESUNAVAIL) condition.	"Exit XPCERES" on page 233
XPCFTCH	Program control program	Before an application program is given control.	"Exit XPCFTCH" on page 242
XPCHAIR	Program control program	Before a HANDLE ABEND routine is given control.	"Exit XPCHAIR" on page 244
XPCREQ	Program control program	Before a LINK request is processed.	"Exit XPCREQ" on page 232
XPCREQC	Program control program	After a LINK request has been completed.	"Exit XPCREQC" on page 235
XPCTA	Program control program	After an abend occurs and before the environment is modified.	"Exit XPCTA" on page 246
XRCINIT	User log record recovery program	During warm and emergency restart, if user recovery log records are detected in the CICS system log.	"Exit XRCINIT" on page 313
XRCINPT	User log record recovery program	During warm and emergency restart, for each user recovery log record found in the CICS system log.	"Exit XRCINPT" on page 314
XRMIIN	Resource manager interface program	Before an EXEC DLI, EXEC SQL, or RMI command runs.	"Exit XRMIIN" on page 250
XRMIOUT	Resource manager interface program	After an EXEC DLI, EXEC SQL, or RMI command runs.	"Exit XRMIIN" on page 250
XRSINDI	Resource management modules	Immediately after successfully installing or discarding a resource.	"Resource management installation and discard exit XRSINDI" on page 252
XSNEX	Security manager domain	Restore old CICS sign-on and sign-off behavior (pre-CICS TS 2.1)	"Exit XSNEX" on page 262
XSNOFF	Security manager domain	After a terminal user signs off.	"Exit XSNOFF" on page 261
XSNON	Security manager domain	After a terminal user signs on.	"Exit XSNON" on page 260
XSRAB	System recovery program	When the system recovery program finds a match for an MVS abend code in the SRT.	"System recovery program exit XSRAB" on page 265
XSTERM	System termination program	During a normal system shutdown, immediately before TD buffers are cleared.	"System termination program exit XSTERM" on page 269
XSTOUT	Statistics domain	Before a statistics record is written to SMF.	"Exit XSTOUT" on page 264

Table 3. Alphabetical list of global user exit points (continued)

Exit name	Module or domain	Where or when invoked	Topic
XSZARQ	Front End Programming Interface	After a FEPI request has completed.	"Front End Programming Interface exits XSZARQ and XSZBRQ" on page 89
XSZBRQ	Front End Programming Interface	Before a FEPI request is actioned.	"Front End Programming Interface exits XSZARQ and XSZBRQ" on page 89
XTCATT	Terminal control program	Before task attach.	"Exit XTCATT" on page 289
XTCIN	Terminal control program	After an input event.	"Exit XTCIN" on page 288
XTCOUT	Terminal control program	Before an output event.	"Exit XTCOUT" on page 288
XTDEREQ	Transient data EXEC interface program	Before CICS processes a transient data API request.	"Exit XTDEREQ" on page 304
XTDEREQC	Transient data EXEC interface program	After a transient data API request has completed.	"Exit XTDEREQC" on page 305
XTDIN	Transient data program	After receiving data from QSAM (extrapartition) or VSAM (intrapartition).	"Exit XTDIN" on page 301
XTDOUT	Transient data program	Before passing data to a QSAM (extrapartition) or VSAM (intrapartition) user-defined transient data queue.	"Exit XTDOUT" on page 302
XTDREQ	Transient data program	Before request analysis.	"Exit XTDREQ" on page 300
XTSEREQ	Temporary storage EXEC interface program	Before CICS processes a temporary storage API request.	"Exit XTSEREQ" on page 277
XTSEREQC	Temporary storage EXEC interface program	After a temporary storage API request has completed.	"Exit XTSEREQC" on page 278
XTSPTIN	Temporary storage domain	Before invocation of a TSPT function.	"Exit XTSPTIN" on page 273
XTSPTOUT	Temporary storage domain	After invocation of a TSPT function.	"Exit XTSPTOUT" on page 275
XTSQRIN	Temporary storage domain	Before invocation of a TSQR function.	"Exit XTSQRIN" on page 270
XTSQROUT	Temporary storage domain	After invocation of a TSQR function.	"Exit XTSQROUT" on page 272

Table 3. Alphabetical list of global user exit points (continued)

Exit name	Module or domain	Where or when invoked	Topic
XWBAUTH	Web domain	During processing of an **EXEC CICS WEB SEND** or **EXEC CICS WEB CONVERSE** command.	"HTTP client open and send exits: XWBAUTH, XWBOPEN and XWBSNDO" on page 164
XWBOPEN	Web domain	During processing of an **EXEC CICS WEB OPEN** or **EXEC CICS INVOKE SERVICE** command.	"HTTP client open and send exits: XWBAUTH, XWBOPEN and XWBSNDO" on page 164
XWBSNDO	Web domain	During processing of an **EXEC CICS WEB SEND** or **EXEC CICS WEB CONVERSE** command.	"HTTP client open and send exits: XWBAUTH, XWBOPEN and XWBSNDO" on page 164
XWSPRROI	Pipeline domain	After any instance of the XWSPRRWI exit is invoked and before the Web services provider business application.	"Exit XWSPRROI" on page 216
XWSPRROO	Pipeline domain	After the Web service provider application returns and before CICS creates the body of the response message.	"Exit XWSPRROO" on page 217
XWSPRRWI	Pipeline domain	After CICS has converted the Web services request body into a language structure and before any instance of the XWSPRROI exit is invoked.	"Exit XWSPRRWI" on page 215
XWSPRRWO	Pipeline domain	After any instance of the XWSPRROO exit and before CICS creates the body of the response message.	"Exit XWSPRRWO" on page 219
XWSRQROI	Pipeline domain	After CICS has processed the outbound Web service response and before any instance of the XWSRQRWI exit.	"Exit XWSRQROI" on page 223
XWSRQROO	Pipeline domain	After any instance of the XWSRQRWO exit has been processed and before the data flows outbound on the Web services transport.	"Exit XWSRQROO" on page 222
XWSRQRWI	Pipeline domain	After CICS has processed the outbound Web service response and after any instance of the XWSRQROI exit.	"Exit XWSRQRWI" on page 224
XWSRQRWO	Pipeline domain	After CICS has converted the application's language structure into a Web services request body and before CICS processes the optional XWSRQROO exit point.	"Exit XWSRQRWO" on page 221
XWSSRROI	Pipeline domain	After CICS has processed the outbound Web service response and before any instance of the XWSSRRWI exit.	"Exit XWSSRROI" on page 228
XWSSRROO	Pipeline domain	After any instance of the XWSSRRWO exit has been processed and before the encryption of data flowing outbound on the Web services transport.	"Exit XWSSRROO" on page 227
XWSSRRWI	Pipeline domain	After CICS has processed the outbound Web service response and after any instance of the XWSSRROI exit.	"Exit XWSSRRWI" on page 229

Table 3. Alphabetical list of global user exit points (continued)

Exit name	Module or domain	Where or when invoked	Topic
XWSSRRWO	Pipeline domain	After CICS has converted the application's language structure into a Web services request body and before CICS processes the optional XWSSRRWO exit point, and before being encrypted by the pipeline's security handler.	"Exit XWSSRRWO" on page 225
XXDFA	DBCTL interface control program	In the active CICS when CICS-DBCTL connection fails.	"DBCTL interface control program exit (XXDFA)" on page 54
XXDFB	DBCTL tracking program	In the alternate CICS when DBCTL fails.	"Exit XXDFB" on page 55
XXDTO	DBCTL tracking program	In the alternate CICS when active DBCTL fails.	"Exit XXDTO" on page 56
XXMATT	Transaction manager domain	When a user transaction is attached.	"Transaction manager domain exit XXMATT" on page 298
XXRSTAT	XRF request processing program	After a z/OS Communications Server failure or a predatory takeover.	"Exit XXRSTAT" on page 334
XZCATT	z/OS Communications Server terminal management program	Before task attach.	"SNA LU management program exit (XZCATT)" on page 315
XZCIN	z/OS Communications Server working set module	After an input event.	"Exit XZCIN" on page 316
XZCOUT	z/OS Communications Server working set module	Before an output event.	"Exit XZCOUT" on page 317
XZCOUT1	z/OS Communications Server working set module	Before a message is broken into RUs.	"Exit XZCOUT1" on page 318
XZIQUE	z/OS Communications Server working set module	1. When an allocate request for a session is about to be queued. 2. When an allocate request succeeds following previous suppression of queuing.	"XZIQUE exit for managing MRO and APPC intersystem queues" on page 318

Global user exit points (by function)

The exit points are grouped according to their functional relationships.

Grouping is generally based on the CICS module or domain in which the exit points occur. However, where exit points in different modules serve a similar function, the exits are grouped under a generic name. The groups of exits are presented in alphabetical order of module or generic name.

The following information is provided for each global user exit point:

- Exit identifier

- Exit location
- DFHUEPAR parameters, if any, that are unique to the exit
- Valid return codes
- XPI calls that can be invoked.

Activity keypoint program exit (XAKUSER)

The XAKUSER exit is invoked during the activity keypointing process. You can use this exit to record, on the system log, user data that must be restored following an emergency restart.

For best performance, journal control requests should not specify WAIT. CICS will force the records by writing a synchronous end of keypoint record upon return from the exit program.

Your exit program should be translated with the NOEDF option. Any program it links to should also be translated with this option. It is not possible to link to programs written in PL/I.

To ensure that it is called during every keypoint, your exit program should be enabled by means of a first phase PLTPI program - see “Writing initialization programs” on page 409. However, if it enabled at this stage, your program should not attempt to link to any program coded in COBOL or C, as it might be invoked before these are initialized.

Note: Your exit program forms part of a critical CICS system activity. If it fails, CICS terminates. Only the listed EXEC CICS commands are allowed in the XAKUSER exit. The exit should link only to other programs with the same restrictions.

Exit XAKUSER

When invoked

During the activity keypointing process.

Exit-specific parameters

UEPAKTYP

Address of a 1-byte field indicating the type of keypoint for which the exit is invoked. The possible values are:

UEPAKPER

Activity keypoint

UEPAKWSD

Warm shutdown keypoint.

Return codes

UERCNORM

Continue processing.

XPI calls

XPI must not be used.

API and SPI calls

The following commands are supported:

- ADDRESS CWA
- ADDRESS EIB

- LINK (but only to local programs; distributed program links may not be used).
- RETURN
- WRITE JOURNALNAME.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Application association data exit in the AP domain (XAPADMGR)

Use the XAPADMGR exit for distributed transactions. XAPADMGR allows you to add user information to the association data of a task, at the point of origin of the distributed transaction. This information could be used later, for example, as a search key for processing carried out through CICSPlex SM.

The exit program is called, if enabled, at the attach of nonsystem tasks for which no input Origin Descriptor Record is provided.

On input, the exit program is passed the association data of the task. The exit might find other relevant information, for inclusion in the association data, from other sources, using CICS commands.

Note: Distributed transactions that use DPL over IPIC connections pass their transaction group ID and origin data, including the user correlator, to be inherited by the mirror task in the target region.

The exit program could perform other activities, such as logging of information found in the association data, for purposes such as auditing or accounting of workloads. For more information on association data and origin data, see Association data in Getting started.

Exit XAPADMGR

When called

At the attach of a nonsystem task that has no inherited association data passed to it.

Exit-specific parameters

UEPADCB

Address of the selectable association data control block. This is mapped by the DFHMNADS DSECT.

UEPADCBL

Length, in bytes, of the association data control block.

UEPUCD
Address of a 64-byte output area in which the exit program can place the user correlation data.

Return codes

UERCNORM
Continue processing.

XPI calls
All can be used.

API and SPI calls
All can be used, except for:
- EXEC CICS ABEND
- EXEC CICS PERFORM SHUTDOWN

Sample exit program, DFH$APAD

CICS provides a sample global user exit program, DFH$APAD, for use at the XAPADMGR exit point. The exit program is called, if enabled, when nonsystem tasks for which no input Origin Descriptor Record is provided are attached.

DFH$APAD performs the following processing:
- Provides addressability to the association data provided as input to the exit.
- Chooses a field from this data and places it in the output buffer.
- Adds a field to the user correlation data in the output buffer.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

"Application associated data sample exit program (DFH$APAD)" on page 22
CICS includes a sample global user exit program for the application associated data exit. The sample program is DFH$APAD.

Basic mapping support exits (XBMIN, XBMOUT)

Two basic mapping support exits are provided: XBMIN and XBMOUT. The XBMIN exit allows you to intercept a RECEIVE MAP request after BMS has successfully processed the request. The XBMOUT exit allows you to intercept a SEND MAP request after BMS has successfully processed the request; or, if cumulative mapping is in progress, on completion of each page of output.

The XBMIN exit is called, if enabled, when all the following are true:
- A RECEIVE MAP command has been successfully processed.

- The map referenced in the command contains at least one field specified as VALIDN=USEREXIT.
- At least one USEREXIT field has been returned in the inbound datastream and has been mapped into the application data structure.

Using XBMIN, you can:

- Analyze each field defined as VALIDN=USEREXIT mapped to the application on this request
- Use the mapset name, map name, and field length defined in the map, and the actual length of field data returned in the inbound datastream
- Modify the data in each field.

The XBMOUT exit is called, if enabled, when all the following are true:

- A SEND MAP command has been successfully processed.
- The map referenced in the command contains at least one field specified as VALIDN=USEREXIT.
- At least one USEREXIT field has been generated in the outbound datastream.

Using XBMOUT, you can:

- Analyze each field defined as VALIDN=USEREXIT which has been generated in the outbound datastream
- Use the mapset name, map name, and field length defined in the map, and the actual length of field data placed in the outbound datastream
- Modify the data in each field
- Modify the attributes sent with each field.

Both exits are passed four exit-specific parameters:

1. The address of the TCTTE associated with the mapping request
2. The address of the system EIB associated with the task issuing the mapping request
3. The address of a halfword binary count of the number of elements in the *field element table*
4. The address of the field element table.

Sample program, DFH$BMXT

CICS supplies a sample program, DFH$BMXT, that shows how mapped input and output data can be modified with reference to the information provided in the "field element" table. A copybook, DFHXBMDS, is also supplied. This copybook is a DSECT which defines the structure of the field element.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

"Basic mapping support sample exit program (DFH$BMXT)" on page 23
CICS includes a sample global user exit program for the basic mapping support user exits. The sample program is DFH$BMXT.

Exit XBMIN

This exit is invoked after basic mapping support (BMS) has successfully processed an input mapping operation.

When invoked

After BMS has successfully processed an input mapping operation.

Exit-specific parameters

UEPBMTCT
Address of the TCTTE associated with the mapping request.

UEPEXECB
Address of the system EIB associated with the task.

UEPBMCNT
Address of the halfword binary number of "field elements" in the field element table.

UEPBMTAB
Address of the field element table.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls

All can be used.

Exit XBMOUT

This exit is invoked after basic mapping support (BMS) has successfully completed a page of output during an output mapping operation.

When invoked

After BMS has successfully completed a page of output during an output mapping operation.

Exit-specific parameters

UEPBMTCT
Address of the TCTTE associated with the mapping request.

UEPEXECB
Address of the system EIB associated with the task.

UEPBMCNT
Address of the halfword binary number of "field elements" in the field element table.

UEPBMTAB
Address of the field element table.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

The field element table structure

The *field element table* contains one or more elements which provide information about each "field of interest" passed to the exit.

A "field of interest" is a field which has been defined as VALIDN=USEREXIT in the map source file used to create the mapset referenced in the mapping operation.

Each field element has the following structure:

BMXMAPST
is an 8-byte area which contains the name of the mapset associated with this field. If terminal or alternate suffixes are used with mapset names in your CICS installation, the mapset name may have a suffix appended to the name specified in the mapping request.

BMXMAP
is a 7-byte area which contains the name of the map associated with this field.

BMXFDFB
is a one-byte field copied from the field specification in the map load module. It contains indicators as follows:

X'80' CASE=MIXED

X'40' Group field entry

X'20' Group field descriptor

X'10' ATTRB=DET

X'08' JUSTIFY=ZERO

X'04' JUSTIFY=RIGHT

X'02' INITIAL,XINIT, or GINIT specified

X'01' Named field (DSECT entry exists)

BMXMAPLN
is a halfword binary value which contains the field length defined in the LENGTH option of the DFHMDF macro.

BMXACTLN
is a halfword binary value which contains the actual length of the data received or transmitted in this field.

BMXDATA
is the address of the field data.

In the XBMIN exit, BMXDATA points into a work area which BMS has obtained for input mapping purposes. When the exit returns control, this work area is copied to the application data structure associated with this map.

In the XBMOUT exit, BMXDATA points into a terminal input/output area (TIOA) in which BMS has generated an output datastream. When the exit

returns control, the TIOA is disposed of in accordance with the disposition of the TERMINAL (the default), SET, or PAGING option specified on the SEND MAP request.

BMXATTR
is only relevant in the XBMOUT exit. It is the address of the attributes (if any) which BMS has placed in the output datastream preceding this field.

BMXMAPOF
is the offset of the field in the map. For example, if a map is defined as

```
MYMAP  DFHMDI SIZE=(12,40)
```

and a field in this map is defined as

```
FLDA   DFHMDF POS=(5,1)
```

the offset of this field (relative to zero) is 160 in decimal notation. In this example, BMXMAPOF would contain the value X'00A0'.

BMXBUF
is the offset of the field in the device buffer. Usually—that is, when the map dimensions are the same as the current screensize in use by the device—this value will be the same as that of BMXMAPOF. However, using the example given in the BMXMAPOF description, if MYMAP is sent to a device currently using a 24 by 80 screensize, the offset of the field in the device buffer (again relative to zero) is 320 in decimal notation. In this example, BMXBUF would contain the value X'0140'.

Programming the XBMIN exit

When programming the XBMIN exit it is important to consider data length.

The actual data length (in BMXACTLN) might be less than the length defined in the map (in BMXMAPLN). This could happen, for example, if a terminal operator does not completely fill a data entry field. In this case, BMS will have right- or left-justified the data in the field and padded the field with blank or zero characters. This justification and padding occurs before the exit is invoked. Your exit program can, by checking the bit settings in the BMXFDFB field, determine how BMS performed justification and padding for the field.

The actual data length (in BMXACTLN) might be greater than the length defined in the map (in BMXMAPLN). This could happen, for example, if a map contains an unprotected field which is not immediately followed by another field. This allows the terminal operator to enter data past the end of the field. When this occurs, the data field is truncated by BMS according to the length defined for the field in the map. However, BMXACTLN contains the length of data found in the inbound datastream.

When modifying data in the XBMIN exit, the safest method is to use the length provided in BMXMAPLN, but to ensure that any pad characters added by BMS are preserved.

BMXATTR must be ignored in the XBMIN exit; it always contains binary zeroes.

Programming the XBMOUT exit

When programming the XBMOUT exit it is important to consider the actual data length.

The actual data length (in BMXACTLN) may be less than the length defined in the map (in BMXMAPLN). This occurs due to the compression of trailing nulls performed by BMS for each output field.

The actual length of data cannot be changed in the exit program. The exit is invoked after the output datastream has been generated; consequently, an attempt to alter the data length could result in an invalid datastream. Therefore, if an XBMOUT exit program modifies data, it must do so with reference to the length value in BMXACTLN.

BMXDATA may contain a null value. This can be caused by a SEND MAP request with the MAPONLY option when the map has fields without default data; this causes BMS to send an attribute sequence for the field but no data.

BMXATTR may contain a null value. This can be caused by a SEND MAP request with the DATAONLY option, when the application is updating the data in a field and not the attributes.

Cumulative mapping operations:

When an application is performing cumulative mapping—that is, issuing a sequence of SEND MAP commands with the ACCUM option—BMS builds composite display in which a single page of output might be constructed from multiple SEND MAP requests.

When cumulative mapping occurs, the XBMOUT exit is called when a page has been built, not as each SEND MAP request is processed.

Message routing:

When an application builds a routing message—for example, it issues a ROUTE command followed by one or more SEND MAP commands with the SET or PAGING option specified—the XBMOUT exit is invoked in the same way as for a non-routed mapping request.

However, the UEPBMTCT parameter is passed as a null value for a routed message. This is because a routed message may be destined for multiple devices, and BMS has optimized the features supported by the devices targeted by the routed message. When processing a routed message in the XBMOUT exit, referencing the TCTTE for any of these devices would probably not be relevant.

Bridge facility exit XFAINTU

The bridge facility exit is called just after a new bridge facility has been built and just before the bridge facility is deleted.

The bridge facility might be deleted at the end of a task, when zero keep time is specified, or when a keep time expires before the facility is reused.

Exit XFAINTU

When invoked

Just after a bridge facility is created and just before it is deleted.

Exit-specific parameters

UEPFAREQ
Address of a 1-byte field that indicates why the exit has been called. Possible values are:

UEPFAIN
Initialization.

UEPFATU
Tidy-up.

UEPFATUT
Address of a 1-byte field that indicates the type of tidy-up required. Possible values are:

UEPFANTU
Normal tidy-up.

UEPFAETU
Expired tidy-up.

UEPFANAM
Address of the bridge facility name.

UEPFATYP
Address of a 1-byte field that indicates the facility type. The value is always:

UEPFABR
3270 bridge facility.

UEPFAUAA
Address of the bridge facility user area (TCTUA).

UEPFAUAL
Address of a one-byte field containing the length of the bridge facility user area.

UEPFATK
Address of the 8–byte facilitytoken.

UEPFAMCH
Address of a 1-byte field that indicates the mechanism used to start the bridged transaction using this bridge facility. Possible values are:

UEPFASTA
Started using START BREXIT.

UEPFALNK
Started using a link to DFHL3270.

UEPFAREG
Address of a 1-byte field that indicates whether the region owns the bridge facility, or whether it is remote. A bridge facility is owned by the AOR, where it is local, and is remote to the router region. Note that XFAINTU can be called twice in the same region if the AOR and the router are the same region. Possible values are:

UEPFAROU
This region is the router for this bridge facility.

UEPFAAOR
This region is the AOR for this bridge facility.

Return codes

UERCNORM
Continue processing.

XPI calls
All can be used, except those that use recoverable resources.

API calls
All can be used except those that invoke task-related user exits or use recoverable resources.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Data tables management exits XDTRD, XDTAD, and XDTLC

Data tables management exits apply to both CICS shared data tables and CICS coupling facility data tables.

XDTRD and XDTAD allow you to control the selection of records for inclusion in a data table, XDTRD being used to make such selections during loading, and XDTAD being invoked when records are subsequently added to a loaded data table (or to a CFDT that did not require loading). XDTRD also allows the contents of records that are included in a user-maintained table, or a coupling facility data table, to be modified before they are added.

For CICS shared data tables, XDTLC enables you to take action based on the fact that a data table has completed loading, which might be to end some restrictions that you have decided to place on access to the data table during loading, or to cater for an unsuccessful completion of the loading.

For a coupling facility data table, XDTLC allows your global user exit program to decide whether to accept an unsuccessfully loaded coupling facility data table. If the user exit program decides to accept the table, it remains open and available for access, but CICS does not mark it as loading completed. This is also the default action if no XDTLC exit is enabled. This means that application programs continue to receive the LOADING condition for any records that are beyond the key range of records successfully loaded into the table. This ensures that application programs are aware that not all the expected data is available. It also allows you to retry the load, when the cause of the failure has been corrected, by closing the file that initiated the load and reopening it. Alternatively, you could open another load-capable file that refers to the same data table. If your exit program decides to reject the table, it is closed and the records already loaded remain in the table. If the cause of the failure is corrected, a subsequent open for the data table allows the load to complete. XDTLC is not invoked for a coupling facility data table that is not loaded from a source data set.

Note that a program invoked from any of these exit points must declare a DSECT defining the data tables user exit parameter list pointed to by field UEPDTPL. (Although UEPDTPL is defined by a DFHUEXIT call, the parameter list that it addresses is not.) To do this, your program can include the copybook DFHXDTDS, which defines the DT_UE_PLIST DSECT.

If any tables specify OPENTIME=STARTUP or are opened implicitly, you should provide a program list table post-initialization (PLTPI) program to activate the user exits. Otherwise, the data table might start loading before the exits can be enabled. For more details about PLTPI programs, see Chapter 5, "Writing initialization and shutdown programs," on page 409.

Note: For additional information about using these exits with CICS shared data table support, see Shared data tables overview in Shared Data Tables.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

"Data tables sample exit programs (DFH$DTAD, DFH$DTLC, DFH$DTRD)" on page 23
CICS includes three sample global user exit programs, one for each of the data tables exit points. The sample programs are DFH$DTAD, DFH$DTLC, DFH$DTRD.

Exit XDTRD

The XDTRD user exit is invoked just before CICS attempts to add to the data table a record that has been retrieved from the source data set.

This normally occurs when the loading process retrieves a record during the sequential copying of the source data set. However, it can also occur when an application retrieves a record that is not in the data table and:

- For a user-maintained data table, loading is still in progress, or
- For a CICS-maintained data table, loading terminated before the end of the source data set was reached (because, for example, the data table was full).

Note: For a coupling facility data table the XDTRD exit is invoked only for a table that is loaded from a source data set.

The record retrieved from the source data set is passed as a parameter to the user exit program—see fields UEPDTRA and UEPDTRL. Your exit program can choose (depending, for example, on the key value—see fields UEPDTKA and UEPDTKL) whether to include the record in the data table or not.

Alternatively, the exit program can request that all subsequent records up to a specified key are skipped—see field UEPDTSKA; these records are not passed to the exit program. This facility is available only during loading. You can specify the key as a complete key, or you can specify just the leading characters by padding the skip-key area with binary zeros.

For a user-maintained data table, the program can also modify the data in the record to reduce the storage needed for the data table. Application programs that use the data table must be aware of any changes made to the record format by the exit program. If the record length is changed, the exit program must set the new length in the parameter list—see field UEPDTRL. The new length must not exceed the data buffer length—see field UEPDTRBL.

When invoked

Just before CICS tries to add to the data table a record that has been retrieved from the source data set.

Exit-specific parameters

UEPDTPL

Address of the data table user exit parameter list, which is mapped by DSECT DT_UE_PLIST in copybook DFHXDTDS. The data table user exit parameter list contains:

UEPDTNAM

The 8-character data table name.

UEPDTFLG

A 1-byte flag field. The possible bit settings are:

UEPDTSDT (X'80')

The exit has been invoked by CICS shared data table support.

UEPDTCMT (X'40')

This is a CICS-maintained table. Only meaningful if UEPDTSDT is on.

UEPDTOPT (X'20')

The exit has been invoked for table loading. This means that optimization by skipping can be requested.

UEPDTCFT(X'10')

The exit has been invoked by coupling facility data table support.

UEPDTUMT (X'08')

This is a user-maintained table. Only meaningful if UEPDTSDT is on.

UEPDTRA

The address of the data record.

UEPDTRBL

The fullword length of the data table buffer.

UEPDTRL

The fullword length of the data record.

For user-maintained tables, the exit program can set a new length in this field, if it amends the record.

UEPDTKA
The address of the data table key.

UEPDTKL
The fullword length of the data table key.

UEPDTDSL
The fullword length of the name of the source data set. Only meaningful if either UEPDTSDT or UEPDTCFT is on.

UEPDTDSN
A 44-character field containing the name of the source data set. Only meaningful if either UEPDTSDT or UEPDTCFT is on.

UEPDTSKA
The address of a skip-key area. When invoked for table loading, your exit program can return a key of length UEPDTKL in this area, and request load optimization by setting a return code of UERCDTOP. Only meaningful if either UEPDTSDT or UEPDTCFT is on.

Return codes

UERCDTAC
Add the record to the data table.

UERCDTRJ
Reject the record: that is, do not add it to the table.

UERCDTOP
Skip this and the following records until a key is found that is equal to or greater than the key specified in the skip-key area. Only meaningful if either UEPDTSDT or UEPDTCFT is on.

XPI calls
All can be used.

Exit XDTAD

Exit XDTAD is invoked when a write request is issued to a data table.

For a user-maintained data table and coupling facility data table, the user exit is invoked once - before the record is added to the data table. For a CICS-maintained data table, the user exit is invoked twice - before the record is added to the source data set and then again before the record is added to the data table.

The record written by the application is passed as a parameter to the user exit program - see fields UEPDTRA and UEPDTRL. Your exit program can choose (depending on the key value, for example see fields UEPDTKA and UEPDTKL) whether to include the record in the data table or not. This decision is indicated by setting the return code.

The XDTAD exit must not modify the data in the record. If you used XDTRD to truncate the data records when the data table was loaded, you must code your application so that it only tries to write records of the correct format for the data table.

A sample XDTAD exit program is listed in Shared data tables overview in Shared Data Tables.

When invoked

One or more times during the processing of a write request to a data table.

Exit-specific parameters

UEPDTPL

Address of the data table user exit parameter list, which is mapped by DSECT DT_UE_PLIST in copybook DFHXDTDS. The data table user exit parameter list contains:

UEPDTNAM

The 8-character data table name.

UEPDTFLG

A 1-byte flag field. The possible bit settings are:

UEPDTSDT (X'80')

The exit has been invoked by CICS shared data table support.

UEPDTCMT (X'40')

This is a CICS-maintained table. Only meaningful if UEPDTSDT is on.

UEPDTCFT(X'10')

The exit has been invoked by coupling facility data table support.

UEPDTUMT (X'08')

This is a user-maintained table. Only meaningful if UEPDTSDT is on.

UEPDTRA

The address of the data record.

UEPDTRBL

The fullword length of the data table buffer.

UEPDTRL

The fullword length of the data record.

UEPDTKA

The address of the data table key.

UEPDTKL

The fullword length of the data table key.

UEPDTDSL

The fullword length of the name of the source data set. Only meaningful if either UEPDTSDT or UEPDTCFT is on.

UEPDTDSN

A 44-character field containing the name of the source data set. Only meaningful if either UEPDTSDT or UEPDTCFT is on.

Return codes

UERCDTAC

Add the record to the data table.

UERCDTRJ

Reject the record: that is, do not add it to the table.

XPI calls

All can be used.

Exit XDTLC

The XDTLC user exit is invoked at the completion of data table loading—whether successful or not. The user exit is not invoked if the data table is closed for any reason before loading is complete. The XDTLC exit is invoked for a coupling facility data table only if the table is loaded from a source data set.

The exit program is informed if the loading did not complete successfully—see field UEPDTORC. This could occur, for example, if the maximum number of records was reached or there was insufficient virtual storage. In this case, the exit program can request that the file is closed immediately, by setting the return code.

When invoked

At the completion of table loading. It is not invoked if the loading process was terminated because the data table had been closed.

Exit-specific parameters

UEPDTPL

Address of the data table user exit parameter list, which is mapped by DSECT DT_UE_PLIST in copybook DFHXDTDS. The data table user exit parameter list contains:

UEPDTNAM

The 8-character data table name.

UEPDTFLG

A 1-byte flag field. The possible bit settings are:

UEPDTSDT (X'80')

The exit has been invoked by CICS shared data table support.

UEPDTCMT (X'40')

This is a CICS-maintained table. Only meaningful if UEPDTSDT is on.

UEPDTCFT(X'10')

The exit has been invoked by coupling facility data table support.

UEPDTUMT (X'08')

This is a user-maintained table. Only meaningful if UEPDTSDT is on.

UEPDTORC

Data table open result code. The possible values are:

UEPDTLCS

Load successful

UEPDTLFL

Load unsuccessful.

UEPDTDSL

The fullword length of the name of the source data set. Only meaningful if either UEPDTSDT or UEPDTCFT is on.

UEPDTDSN
A 44-character field containing the name of the source data set. Only meaningful if either UEPDTSDT or UEPDTCFT is on.

Return codes

UERCDTOK
Accept the data table in its present state

UERCDTCL
Close the data table.

XPI calls
All can be used.

DBCTL interface control program exit (XXDFA)

This exit is invoked by an active CICS if its connection to DBCTL fails.

When invoked
By an active CICS when its connection to DBCTL fails. Your exit program is invoked after the active CICS has informed the alternate CICS of the failure.

Exit-specific parameters

UEPDBXR
Address of CICS XRF information for use with DBCTL. The CICS XRF information can be mapped using the DSECT DFHDXUEP.

Return codes

UERCNOAC
Take no action.

UERCSWCH
Switch to the alternate DBCTL.

UERCABNO
Abend CICS without a dump.

UERCABDU
Abend CICS with a dump.

XPI calls
TRANSACTION_DUMP must not be used.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

DBCTL tracking program exits (XXDFB, XXDTO)

These exits are invoked if the connection to DBCTL fails, or if CICS performs takeover.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XXDFB

The XXDFB exit is invoked when a message is received from the active CICS indicating that the connection to DBCTL failed.

When invoked

By the alternate CICS when it receives a message from the active CICS indicating that connection to DBCTL has failed. The alternate and active CICS systems are running in different MVS images, perhaps in different central processing complexes (CPCs). More information about these exits, see Overview of Database Control (DBCTL) in Product overview.

Exit-specific parameters

UEPDBXR
Address of CICS XRF information for use with DBCTL. The CICS XRF information can be mapped using the DSECT DFHDXUEP.

Return codes

UERCNOAC
Take no action.

UERCSWCH
Switch to the alternate DBCTL.

UERCABNO
Abend CICS without a dump.

UERCABDU
Abend CICS with a dump.

The return code 'UERCNORM' is not available for use at this exit point.

XPI calls

The following must not be used:

- INQUIRE_MONITORING_DATA
- MONITOR
- TRANSACTION_DUMP
- WRITE_JOURNAL_DATA.

Exit XXDTO

Exit XXDTO is invoked by an alternate CICS when it performs takeover.

When invoked

By an alternate CICS when it performs takeover under the following conditions:

- The active and alternate CICS systems are in different MVS images, perhaps in different processors.
- The active CICS was connected to, or trying to connect to, a DBCTL subsystem. (This does not include disconnecting from one DBCTL and reconnecting to another.)
- The takeover was not initiated by the XXDFB exit, or the takeover was initiated by XXDFB but the active system reestablished a DBCTL connection before takeover occurred and XXDTO was driven for a new DBCTL takeover decision.

Exit-specific parameters

UEPDBXR
Address of CICS XRF information for use with DBCTL. The CICS XRF information can be mapped using the DSECT DFHDXUEP.

Return codes

UERCNOAC
Take no action.

UERCSWCH
Switch to the alternate DBCTL.

UERCABNO
Abend CICS without a dump.

UERCABDU
Abend CICS with a dump.

The return code UERCNORM is not available for use at this exit point.

XPI calls

The following must not be used:

- INQUIRE_MONITORING_DATA
- MONITOR
- TRANSACTION_DUMP
- WRITE_JOURNAL_DATA.

Dispatcher domain exits XDSBWT and XDSAWT

The XDSBWT and XDSAWT exit points are located before and after the operating system wait. You cannot use CICS services in an exit program that is invoked from these exit points.

The XDSBWT and XDSAWT exits can be used to control the swapping state of the CICS address space. However, if the default state of the address space is non-swappable, you cannot use these exits to override this state.

CICS uses a counter that is incremented for every SYSEVENT DONTSWAP request and decremented for every SYSEVENT OKSWAP request down to a minimum of 0. A SYSEVENT DONTSWAP request is issued when this counter goes up from 0 to 1. A SYSEVENT OKSWAP request is issued when this counter goes down from 1 to 0. In all other circumstances, the SYSEVENT is not issued.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XDSBWT

This exit is invoked before an operating system wait issued by the quasi-reentrant CICS TCB.

When invoked

Before an operating system wait issued by the quasi-reentrant CICS TCB.

Exit-specific parameters

None.

Return codes

UERCNORM

Continue processing.

UERCSWAP

Issue SYSEVENT to allow address space swapping.

XPI calls

Must not be used.

Exit XDSAWT

This exit is invoked after an operating system wait issued by the quasi-reentrant CICS TCB.

When invoked

After an operating system wait issued by the quasi-reentrant CICS TCB.

Exit-specific parameters

UEPSYSRC

Address of the 4-byte return code from the SYSEVENT request made before the operating system wait. This return code will be in one of two different forms:

1. The SYSEVENT OKSWAP return code, or
2. If the SYSEVENT request was rejected by CICS, a special CICS return code which will take one of the following decimal values:

   17 The SYSEVENT OKSWAP was not issued. The outstanding count of SYSEVENT OKSWAP requests exceeds the count of SYSEVENT DONTSWAP requests. Before a SYSEVENT OKSWAP can be issued, a SYSEVENT DONTSWAP must be requested.

   19 The SYSEVENT OKSWAP was not issued. The outstanding count of SYSEVENT DONTSWAP requests still exceeds the count of SYSEVENT OKSWAPs. Further SYSEVENT OKSWAPs must be requested before a SYSEVENT OKSWAP is issued by CICS.

Return codes

UERCNORM

Continue processing.

UERCNOSW
Issue SYSEVENT to suppress address-space swapping.

XPI calls
Must not be used.

DL/I interface program exits XDLIPRE and XDLIPOST

The XDLIPRE and XDLIPOST exit points are invoked following the issue of an **EXEC DLI** command or DL/I call. Exit XDLIPRE is invoked before the request is processed and XDLIPOST is invoked after the request is processed.

When the request is function shipped, the exits are invoked from both the application-owning region and the database-owning region. However, there are restrictions when they are invoked in a database-owning region:

1. The descriptions of the exits show the general format of the parameter list of the application. For detailed information about the format of the CALL-level DL/I parameter list, refer to *IMS Application programming: DL/I calls reference*.
2. For all EXEC DLI calls, the parameter list of the application is in assembler language format; that is, the value of the program language byte pointed to by UEPLANG is always UEPASM, and the parameter list pointed to by UEPAPLIST is always in assembler language format. This format is used because all EXEC DLI calls are converted into assembler-language CALL-level requests.

   An EXEC DLI online request is converted by DFHEDP into a CALL-level request for DFHDLI. IMS™ does not deal directly with EXEC-level parameter lists. The first parameter in the CALL parameter list contains the address of the parameter count. The second parameter in the CALL parameter list contains the address of the function. All other parameters are dependent on the function.
3. In an XDLIPRE exit program, you can change the PSB name and the SYSID name. Changing the name helps availability if the originally specified SYSID fails.

   You can change the SYSID in the following ways:
   - A remote value to another remote value
   - The local value to a remote value
   - A remote value to the local value.

   Changing the SYSID has an effect only if the associated PSB has a PDIR entry. The SYSID can be the local CICS (that is, the SYSIDNT specified on the CICS region) or a remote connection name. For the new SYSID to be used, the PSB name must have a PDIR entry; if it does not have a PDIR entry, the assumption is made that the local CICS is connected to DBCTL, and an attempt is made to run the IMS request there. An IMS schedule failure is handled in the same way as a failure to route to a connection that does not exist. If the SYSID is changed to either the same value as the SYSIDNT of the local CICS, or blanks (hex '40404040'), CICS attempts to run the IMS request on the local system.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XDLIPRE

Exit XDLIPRE is invoked on entry to the DL/I interface program.

Programs running in this exit must be coded to threadsafe standards and defined to CICS as threadsafe.

Exit-specific parameters

UEPCTYPE
Address of type-of-request byte. Values are:

UEPCEXEC
The original request was an EXEC DLI request.

UEPCCALL
The original request was a CALL-level request.

UEPCSHIP
The request has been function shipped from another region. When this value is set, restrictions apply to the setting and use of the rest of the exit parameters, as described.

UEPAPLIST
Address of application's parameter list. The general format for COBOL and assembler language is:

```
plist address --> parm1 address --> parm1
                  parm2 address --> parm2
                  parm3 address --> parm3
                  ..............
                  up to a maximum of 18 parameters
                  excluding the optional parmcount.
```

The general format for PL/I is:

```
plist address --> parm1 address --> parm1 (parmcount)
       parm2 address --> locator descriptor --> parm2
       parm3 address --> locator descriptor --> parm3
       ..............
       up to a maximum of 18 parameters
```

When UEPCTYPE is not UEPCSHIP, your exit program can change any of the parameters in the application parameter list. For UEPCSHIP requests, your exit program **cannot** change any of the parameters. Furthermore, for UEPCSHIP requests, UEPAPLIST points to a copy of the parameter list in the above format, but which contains only the first two parameters, `parm1` and `parm2`.

Note: For PL/I applications, `parm1` may or may not contain a parameter-count. Your exit program should check this field before using it.

UEPLANG
Address of program language byte. Values are:

UEPPLI
PL/I

UEPCBL
COBOL

UEPASM
Assembler language.

For UEPCSHIP requests, the language is always assembler.

UEPIOAX
Address of I/O area existence flag byte:

UEPIOA1
I/O area exists.

For UEPCSHIP requests, the I/O area existence flag is always off.

UEPIOA
Address of I/O area. This is the application's IOAREA, or DFHEDP's IOAREA in the case of EXEC DLI. The contents of the IOAREA can be overwritten in the exit: the new contents are used when the DL/I request is processed. However, it should be noted that IOAREAs can be in a program's static storage and, in this case, should not be overwritten.

For UEPCSHIP requests, UEPIOA is always zero.

UEPPSBNX
Address of PSB existence flag byte:

UEPPSB1
A PSB exists.

UEPPSBNM
Address of an area containing the 8-character PSB name. The contents of the area can be overwritten by the exit, for all types of request including UEPCSHIP; the new contents are used when the DL/I request is processed.

UEPSYSDX
Address of the SYSID existence flag byte:

UEPSYS1
A SYSID exists.

UEPSYSID
Address of an area containing the 4-character SYSID name. The contents of the area can be overwritten by the exit, for all types of request including UEPCSHIP; the new contents are used when the DL/I request is processed.

Return codes

UERCNORM
Continue processing

UERCBYP
Bypass DL/I request and return

UERCPURG
Task purged during XPI call.

XPI calls

All can be used.

Exit XDLIPOST

Exit XDLIPOST is invoked on exit from the DL/I interface program.

Programs running in this exit must be coded to threadsafe standards and defined to CICS as threadsafe.

Exit-specific parameters

UEPCTYPE

Address of type-of-request byte. Values are:

UEPCEXEC

An EXEC DLI request.

UEPCCALL

A CALL-level request.

UEPCSHIP

The request has been function shipped from another region. When this value is set, restrictions apply to the setting and use of the rest of the exit parameters, as described.

UEPAPLIST

Address of application's parameter list. The general format for COBOL and assembler language is:

```
plist address --> parm1 address --> parm1
                  parm2 address --> parm2
                  parm3 address --> parm3
                  ..............
                  up to a maximum of 18 parameters
                  excluding the optional parmcount.
```

The general format for PL/I is:

```
plist address --> parm1 address --> parm1 (parmcount)
       parm2 address --> locator descriptor --> parm2
       parm3 address --> locator descriptor --> parm3
       ..............
       up to a maximum of 18 parameters.
```

When UEPCTYPE is not UEPCSHIP, your exit program can change any of the parameters in the application parameter list. For UEPCSHIP requests, your exit program **cannot** change any of the parameters. Furthermore, for UEPCSHIP requests, UEPAPLIST points to a copy of the parameter list in the previous format, but which contains only the first two parameters `parm1` and `parm2`. See also “DL/I interface program exits XDLIPRE and XDLIPOST” on page 58.

Note: For PL/I applications, `parm1` might or might not contain a parameter-count. Your exit program should check this field before using it.

UEPLANG

Address of program language byte. Its values are:

UEPPLI

PL/I

UEPCBL
COBOL

UEPASM
assembler language.

For UEPCSHIP requests, the language is always assembler.

UEPIOAX
Address of I/O area existence flag byte:

UEPIOA1
I/O area exists.

For UEPCSHIP requests, the I/O area existence flag is always off.

UEPIOA
Address of I/O area. This is the application's IOAREA, or DFHEDP's IOAREA in the case of EXEC DLI. The contents of the IOAREA can be overwritten in the exit and are returned to the application program in the new form. However, it should be noted that the application's IOAREA could be in the program's static storage and, in this case, should not be overwritten.

For UEPCSHIP requests, UEPIOA is always zero.

UEPUIBX
Address of UIB existence flag byte:

UEPUIB1
a UIB exists.

UEPUIB
Address of the UIB, which is mapped by DFHUIB in module DFHDBCOP. The contents of the UIB can be overwritten in the exit for all types of request, including UEPCSHIP. The new contents are returned to the application or to the region that function shipped the request.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Example use of global user exit XDLIPRE

You can use the global user exit XDLIPRE to change the PSB name that the application program has scheduled at execution time. You can also use the XDLIPRE exit to change the identity of the SYSID, enabling work to be rerouted from a SYSID that becomes unavailable to one that is available.

This section contains Product-sensitive Programming Interface information.

The following figures show an example of XDLIPRE that you can copy and modify. This example is provided for guidance only. For programming information about global user exits, see “DFHZNEPI TYPE=INITIAL—specifying the default routine” on page 493.

```
*********************************************************************
* This is an example for global user exit XDLIPRE                   *
*                                                                   *
* It is invoked before any DLI call being passed to              *
* the remote or DBCTL processors.                          *
*                                                                   *
* A check is made for the presence of a PSB.                        *
* If not, a normal return is made                                   *
*                                                                   *
* If the PSB is in a predefined table, it is changed to a           *
* different value, and a normal return is made.                     *
*                                                                   *
* If not, set PSB name to blanks and normal return.                 *
*                                                                   *
* In all cases,a trace entry is written describing the action       *
* taken, using TRACE-POINT 384  (hex '0180')                        *
*                                                                   *
*********************************************************************
*                                                                   *
* The first few instructions set up the global user exit            *
* environment, identify the user exit point, prepare for the use of*
* the exit programming interface, and copy in the definitions that *
* are to be used by the XPI function.                               *
*                                                                   *
*********************************************************************
*
         DFHUEXIT TYPE=EP,ID=XDLIPRE    PROVIDE DFHUEPAR PARAMETER
*                                       LIST AND LIST OF EXITID
*                                       EQUATES
*
         DFHUEXIT TYPE=XPIENV           SET UP ENVIRONMENT FOR
*                                       EXIT PROGRAMMING INTERFACE
*                                       MUST BE ISSUED BEFORE ANY
*                                       XPI MACROS ARE ISSUED
```

Figure 1. Example of XDLIPRE user exit to change PSB names 1/6

```
*
         COPY  DFHTRPTY                DEFINE PARAMETER LIST FOR
*                                      USE BY DFHTRPTX MACRO
*
         COPY  DFHSMMCY                DEFINE PARAMETER LIST FOR
*                                      USE BY DFHSMMCX MACRO
*
*****************************************************************
*The following DSECT maps a storage area to be used as work area *
*for the information in the TRACE entry.                         *
*****************************************************************
*
DSA      DSECT                         DSECT FOR GETMAINED STORAGE
         USING DSA,R7
*
RETCODE  DS    F                       store return code
MESSAGEA DS    F                       message address for trace
MESSAGEL DS    F                       message length for trace
MESSAGE  DS    0CL37
OLDPSB   DS    CL8
MESS1    DS    CL21
NEWPSB   DS    CL8
*********************************************************************
*The next  instructions  form  the  normal  start  of a global user *
*exit program, setting the program addressing mode to 31-bit, saving*
*the calling program's registers, establishing base addressing*
*and establishing the addressing of the user exit parameter list.   *
*********************************************************************
*
DLIPR    CSECT
DLIPR    AMODE 31
*
         SAVE (14,12)                  SAVE CALLING PROGRAM'S RGSTRS
*
         LR    R11,R15                 SET UP USER EXIT PROGRAM'S
         USING DLIPR,R11               BASE REGISTER
*
         LR    R2,R1                   SET UP ADDRESSING FOR USER
         USING DFHUEPAR,R2             EXIT PARAMETER LIST -- USE
*                                      REGISTER 2 AS XPI CALLS USE
*                                      REGISTER 1
*
*********************************************************************
*Before issuing an XPI function call, set up addressing to XPI     *
*parameter list.                                                   *
*********************************************************************
*
         L     R5,UEPXSTOR             SET UP ADDRESSING FOR XPI
*                                      PARAMETER LIST
```

Figure 2. Example of XDLIPRE user exit to change PSB names 2/6

```
*********************************************************************
*  Before issuing an XPI function call, you must ensure that register*
*  13 addresses the kernel stack.                                    *
*********************************************************************
*
         L     R13,UEPSTACK             ADDRESS KERNEL STACK
*
*********************************************************************
*  Issue a GETMAIN to get storage for work area                     *
*********************************************************************
*
         USING DFHSMMC_ARG,R5           MAP PARAMETER LIST
*
         DFHSMMCX CALL,                                                X
               CLEAR,                                                  X
               IN,                                                     X
               FUNCTION(GETMAIN),                                      X
               GET_LENGTH(100),                                        X
               STORAGE_CLASS(USER),                                    X
               SUSPEND(NO),                                            X
               OUT,                                                    X
               ADDRESS((R7)),                                          X
               RESPONSE(*),                                            X
               REASON(*)
*
*********************************************************************
*  SET UP THE NORMAL RETURN CODE                                    *
*********************************************************************
*
         LA    R6,UERCNORM
         ST    R6,RETCODE
*
*********************************************************************
*  See if a PSB exists                                              *
*********************************************************************
*
         L     R6,UEPPSBNX              PSB EXISTENCE FLAG
         TM    0(R6),UEPPSB1            PSB EXISTS?
         BO    PSBCALL                  YES
         MVC   MESSAGE,MESS3T           NO-MOVE MESSAGE TO DSA
         B     TRACE
*
*********************************************************************
*  See if we want to change a PSB name                              *
*********************************************************************
*
PSBCALL  EQU   *
         L     R6,UEPPSBNM              ADDRESS OF PASSED PSB NAME
         LA    R8,PSBS                  ADDRESS OF table of PSB pairs
         CLC   0(8,R6),0(R8)            SAME?
```

Figure 3. Example of XDLIPRE user exit to change PSB names 3/6

```
         BE    FOUND                   YES
         LA    R8,16(R8)               BUMP TO NEXT PAIR
         CLC   0(8,R6),0(R8)
         BE    FOUND
         LA    R8,16(R8)               BUMP TO NEXT PAIR
         CLC   0(8,R6),0(R8)
         BE    FOUND
         B     NOTFOUND                NO MATCH - END
*
*********************************************************************
*  Move new PSB name in                                             *
*********************************************************************
*
FOUND    EQU   *
         MVC   0(8,R6),8(R8)
*
*********************************************************************
*  SET UP MESSAGE BLOCK FOR TRACE ENTRY FOR CHANGED NAME            *
*********************************************************************
*
         MVC   MESS1,MESS1T            SET UP MESSAGE
         MVC   NEWPSB,8(R8)            NEW PSB NAME
         MVC   OLDPSB,0(R8)            OLD PSB NAME
         B     TRACE                   GO PUT TRACE ENTRY
*
*********************************************************************
* SET UP MESSAGE BLOCK FOR TRACE ENTRY FOR PSB NOT FOUND            *
* SETUP THE NORMAL RETURN CODE                                      *
*********************************************************************
*
NOTFOUND EQU   *
         MVC   0(8,R6),DUMMYPSB
         MVC   MESS1,MESS2T            SET UP MESSAGE
         MVC   OLDPSB,0(R6)            SUPPLIED PSB NAME
         MVC   NEWPSB,=CL8''           CLEAR FIELD
         LA    R1,UERCNORM             SET UP NORMAL RETURN CODE
         B     TRACE                   GO PUT TRACE ENTRY
*
*********************************************************************
*  Issue trace put macro                                            *
*********************************************************************
*
TRACE    EQU   *
         LA    R6,MESSAGE              STORE ADDRESS...
         ST    R6,MESSAGEA             ...INTO BLOCK DESCRIPTOR
         LA    R6,L'MESSAGE            STORE LENGTH...
         ST    R6,MESSAGEL             ...INTO BLOCK DESCRIPTOR
         LA    R8,384                  SET UP TRACE-ID
*
```

Figure 4. Example of XDLIPRE user exit to change PSB names 4/6

```
         DROP  R5                      REUSE R5 TO MAP DFHTRPT
         USING DFHTRPT_ARG,R5          XPI PARAMETER LIST
*
         DFHTRPTX CALL,                                                X
               CLEAR,                                                  X
               IN,                                                     X
               FUNCTION(TRACE_PUT),                                    X
               POINT_ID((R8)),                                         X
               DATA1(MESSAGEA,MESSAGEL),                               X
               OUT,                                                    X
               RESPONSE(*)
*
*****************************************************************
*When the rest of the exit program is complete, free the storage   *
*and return.                                                       *
*****************************************************************
*
      DROP  R5                   REUSE REGISTER 5 TO MAP DFHSMMC
      USING DFHSMMC_ARG,R5       XPI PARAMETER LIST
*
*****************************************************************
*  Issue the DFHSMMCX macro call                                  *
*  Store the return code in register 6                            *
*****************************************************************
*
          L     R6,RETCODE              PICK UP SAVED RETURN CODE
*
          DFHSMMCX CALL,                                             X
                CLEAR,                                               X
                IN,                                                  X
                FUNCTION(FREEMAIN),                                  X
                ADDRESS((R7)),                                       X
                STORAGE_CLASS(USER),                                 X
                OUT,                                                 X
                RESPONSE(*),                                         X
                REASON(*)
*
*****************************************************************
*Restore registers, set return code, and return to user exit handler*
*****************************************************************
*
          L     R13,UEPEPSA
          ST    R6,16(13)               STORE INTO R15 SLOT OF SA
          RETURN (14,12)
*
*****************************************************************
*old and new PSB names, in pairs                                  *
*****************************************************************
*
```

Figure 5. Example of XDLIPRE user exit to change PSB names 5/6

```
PSBS     EQU   *
          DC    CL8'PC3CONEW'            VALID
          DC    CL8'PC3CONE2'            VALID
          DC    CL8'PC3FRED'             INVALID
          DC    CL8'PC3CONEW'            VALID
          DC    CL8'PC3JOE'              INVALID
          DC    CL8'PC3JOEX'             INVALID
*
MESS1T    DC    CL21' HAS BEEN CHANGED TO '
MESS2T    DC    CL21' WAS NOT FOUND'
MESS3T    DC    CL37'THIS WAS NOT A DLI SCHEDULE CALL'
DUMMYPSB  DC    CL8' '
          LTORG
          END   DLIPR
```

Figure 6. Example of XDLIPRE user exit to change PSB names 6/6

Dump domain exits XDUREQ, XDUREQC, XDUCLSE, and XDUOUT

There are four exits in the dump domain that you can use to capture information before and after a transaction dump or system dump.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

"Dump domain sample exit program (DFH$XDRQ)" on page 23
CICS includes a dump domain sample global user exit program. The sample prom is DFH$XDRQ.

Exit XDUREQ

Exit XDUREQ is invoked immediately before a system or transaction dump is taken.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name, or nulls if there is no current application.

UEPDUMPC
Address of copy of the 8-byte dump code.

UEPABCDE
Address of a copy of the 8-byte Kernel error code in the format xxx/yyyy. xxx denotes the 3-digit hexadecimal MVS completion code (for example 0C1 or D37). If an MVS completion code is not applicable, xxx is three hyphens. The 4-digit code yyyy is a user abend code produced either by CICS or by another product on your system. UEPABCDE is completed only for abend codes corresponding to the following dump codes:

- AP0001
- SR0001
- ASRA
- ASRB
- ASRD

Otherwise this field contains null characters.

UEPDUMPT
Address of the 1-byte dump-type identifier, which contains one of the following values:

UEPDTRAN
A transaction dump was requested.

UEPDSYST
A system dump was requested.

Note: The dump-type identifier indicates the type of dump request that was passed to the dump domain. It does not reflect any modification that may have been made to the original request by a user entry in the dump table.

UEPXDSCP
Address of a 1-byte field indicating the current dump table DUMPSCOPE setting. It contains one of the following values:

UEPXDLOC
A system dump will be taken on the local MVS image only.

UEPXDREL
System dumps will be taken on both the local MVS image, and on related MVS images within the sysplex.

This field may be modified by the exit to update the dump table DUMPSCOPE setting.

UEPXDTXN
Address of a 1-byte field indicating the current dump table TRANDUMP setting. It contains one of the following values:

UEPXDYES
A transaction dump will be taken.

UEPXDNO
A transaction dump will not be taken.

This field may be modified by the exit to update the dump table TRANDUMP setting.

Note: This field is only valid if UEPDUMPT contains the value UEPDTRAN.

UEPXDSYS
Address of a 1-byte field indicating the current dump table SYSDUMP setting. It contains one of the following values:

UEPXDYES
A system dump will be taken.

UEPXDNO
A system dump will not be taken.

This field may be modified by the exit to update the dump table SYSDUMP setting.

UEPXDTRM
Address of a 1-byte field indicating the current dump table SHUTDOWN setting. It contains one of the following values:

UEPXDYES
The CICS system is to shutdown.

UEPXDNO
The CICS system is not to shutdown.

This field may be modified by the exit to update the dump table SHUTDOWN setting.

UEPXDMAX
Address of a 4-byte field which contains the current dump table MAXIMUM setting. This field may be modified by the exit to change the current dump table MAXIMUM setting. A change to the MAXIMUM setting will not suppress this dump request. A return code of UERCBYP may be used to suppress the current dump request.

UEPDXDCNT
Address of a 4-byte field which contains the current dump table CURRENT setting.

UEPXDTST
Address of a 16-byte field which contains the current dump table statistics for this dump code. The addressed field consists of four 4-byte fields containing binary integers:

- Number of transaction dumps taken
- Number of transaction dumps suppressed
- Number of system dumps taken
- Number of system dumps suppressed

Note: Statistics for transaction dumps are valid only if UEPDUMPT contains the value UEPDTRAN.

UEPXDDAE
Address of a 1-byte field which represents the current dump table DAEOPTION setting. It contains one of the following values:

UEPXDYES
The dump is eligible for DAE suppression.

UEPXDNO
The dump will not be suppressed by DAE.

This field may be modified by the exit to update the dump table DAEOPTION setting.

UEPDMPID
Address of a 9-character field in the format xxxx/xxxx, containing the dump identifier. The dump ID is the same as that output by the corresponding dump message.

UEPFMOD
Address of an 8-byte area containing, if the dump code is AP0001, the name of the failing module; otherwise null characters.

Note that field UEPPROG always addresses the name of the *current* application, regardless of where the failure occurred. UEPFMOD addresses the name of the module where the failure occurred, if known.

If the dump code is AP0001, there are three possibilities:

1. The field addressed by UEPFMOD contains the same name as the field addressed by UEPPROG—the failure occured in application code.
2. The field addressed by UEPFMOD contains a different name from the field addressed by UEPPROG—the failure occurred in non-application code.
3. The field addressed by UEPFMOD contains '????????'—the failure was not in application code, but CICS was unable to determine the name of the failing module.

Return codes

UERCNORM
Continue processing.

UERCBYP
Suppress dump.

UERCPURG
Task purged during XPI call.

XPI calls
WAIT_MVS can be used **only** when a UEPDUMPT indicates that a transaction dump is being taken. **Do not use any other calls**.

Related concepts:

"Dump domain sample exit program (DFH$XDRQ)" on page 23
CICS includes a dump domain sample global user exit program. The sample prom is DFH$XDRQ.

Exit XDUREQC

Exit XDUREQC is invoked immediately after a system or transaction dump has been taken, or has failed or been suppressed.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name.

UEPDUMPC
Address of copy of the 8-byte dump code.

UEPABCDE
Address of a copy of the 8-byte Kernel error code in the format xxx/yyyy. xxx denotes the 3-digit hexadecimal MVS completion code (for example X'0C1' or X'D37'). If an MVS completion code is not applicable, xxx is three hyphens. The 4-digit code yyyy is a user abend code produced either by CICS or by another product on your system. UEPABCDE is completed only for abend codes corresponding to the following dump codes:

- AP0001
- SR0001
- ASRA
- ASRB
- ASRD

Otherwise this field contains null characters.

UEPDUMPT
Address of the 1-byte dump-type identifier, which contains one of the following values:

UEPDTRAN
A transaction dump was requested.

UEPDSYST
A system dump was requested.

Note: The dump-type identifier indicates the type of dump request that was passed to the dump domain. It does not reflect any modification that may have been made to the original request by a user entry in the dump table.

UEPXDSCP
Address of a 1-byte field indicating the current dump table DUMPSCOPE setting. It contains one of the following values:

UEPXDLOC
A system dump will be taken on the local MVS image only.

UEPXDREL
System dumps will be taken on both the local MVS image, and on related MVS images within the sysplex.

This field may be modified by the exit to update the dump table DUMPSCOPE setting.

UEPXDTXN
Address of a 1-byte field indicating the current dump table TRANDUMP setting. It contains one of the following values:

UEPXDYES
A transaction dump will be taken.

UEPXDNO
A transaction dump will not be taken.

This field may be modified by the exit to update the dump table TRANDUMP setting.

Note: This field is only valid if UEPDUMPT contains the value UEPDTRAN.

UEPXDSYS
Address of a 1-byte field indicating the current dump table SYSDUMP setting. It contains one of the following values:

UEPXDYES
A system dump will be taken.

UEPXDNO
A system dump will not be taken.

This field may be modified by the exit to update the dump table SYSDUMP setting.

UEPXDTRM
Address of a 1-byte field indicating the current dump table SHUTDOWN setting. It contains one of the following values:

UEPXDYES
The CICS system is to shutdown.

UEPXDNO
The CICS system is not to shutdown.

This field may be modified by the exit to update the dump table SHUTDOWN setting.

UEPXDMAX
Address of a 4-byte field which contains the current dump table MAXIMUM setting. This field may be modified by the exit to change the current dump table MAXIMUM setting.

UEPDXDCNT
Address of a 4-byte field which contains the current dump table CURRENT setting.

UEPXDTST
Address of a 16-byte field which contains the current dump table statistics for this dumpcode. The addressed field consists of four 4-byte fields containing binary integers:

- Number of transaction dumps taken
- Number of transaction dumps suppressed
- Number of system dumps taken
- Number of system dumps suppressed.

Note: Statistics for transactions dumps are valid only if UEPDUMPT contains the value UEPDTRAN.

UEPXDDAE
Address of a 1-byte field which represents the current dump table DAEOPTION setting. It contains one of the following values:

UEPXDYES
The dump was suppressed by DAE.

UEPXDNO
The dump was not suppressed by DAE.

This field may be modified by the exit to update the dump table DAEOPTION setting.

UEPDMPID
Address of a 9-character field in the format xxxx/xxxx, containing the dump identifier. The dump ID is the same as that output by the corresponding dump message.

UEPDRESP
Address of the 2-byte dump response code.

UEPDREAS
Address of the 2-byte dump reason code.

Return codes

UERCNORM
Continue processing.

XPI calls
WAIT_MVS can be used only when a UEPDUMPT indicates that a transaction dump is being taken. Do not use any other calls.

Exit XDUCLSE

This exit is invoked immediately after a transaction dump data set has been closed.

When invoked
Immediately after a transaction dump data set has been closed.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name.

UEPDMPDD
Address of the 8-byte dump data set ddname.

UEPDMPDSN
Address of the 44-byte dump data set dsname.

Return codes

UERCNORM
Continue processing.

UERCSWCH
The autoswitch flag is set on.

XPI calls
WAIT_MVS can be used. **Do not use any other calls**.

Exit XDUOUT

This exit is invoked before a record is written to the transaction dump data set.

When invoked
Before a record is written to the transaction dump data set.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name.

UEPDMPFC
Address of the 1-byte function code. The equated values are:

UEPDMPWR
Buffer is about to be written.

UEPDMPRE
Dump is about to restart after autoswitch.

UEPDMPAB
Abnormal termination of dump.

UEPDMPDY
Buffer is about to be written, and the CICS dump data set is a dummy file or is closed.

UEPDMPBF
Address of the dump buffer, whose length is addressed by the parameter UEPDMPLEN.

UEPDMPLEN
Address of the 2-byte dump-buffer length.

Return codes

UERCNORM
Continue processing.

UERCBYP
Suppress dump record output.

XPI calls
WAIT_MVS can be used. **Do not use any other calls**.

Enqueue EXEC interface program exits XNQEREQ and XNQEREQC

You can use the XNQEREQ exit to intercept enqueue API requests before any action has been taken on the request. You can use the XNQEREQC exit to intercept the response after an enqueue API request has completed.

The API requests affected are:

- `EXEC CICS ENQ`
- `EXEC CICS DEQ`

Using **XNQEREQ**, you can:

- Analyze the API parameter list (function, keywords, argument values, and responses).
- Modify any input parameter value before execution of a request.

- Prevent execution of a request. This enables you to replace the CICS function with your own processing.

Using **XNQEREQC**, you can analyze the API parameter list.

You can also:

- Pass data between your XNQEREQ and XNQEREQC exit programs when they are invoked for the same request
- Pass data between your enqueue exit programs when they are invoked within the same task.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XNQEREQ

This exit is invoked before CICS processes an EXEC CICS ENQ or DEQ request, or attempts to match it to an installed ENQMODEL resource definition.

When invoked

Before CICS processes an EXEC CICS ENQ or DEQ request, or attempts to match it to an installed ENQMODEL resource definition.

Exit-specific parameters

UEPCLPS

Address of a copy of the command parameter list. See "The command-level parameter structure" on page 78.

UEPNQTOK

Address of a 4-byte area which can be used to pass information between XNQEREQ and XNQEREQC for a single enqueue request.

UEPRCODE

Address of a 6-byte hexadecimal copy of the EIB return code EIBRCODE. For details of EIB return codes, see EIB fields, in the *CICS Application Programming Reference* manual.

UEPRESP

Address of a 4-byte binary copy of the EIB response code EIBRESP.

UEPRESP2

Address of a 4-byte binary copy of the EIB response code EIBRESP2.

UEPTSTOK

Address of a 4-byte token which can be used to pass information between successive enqueue requests within the same task (for example, between successive invocations of the XNQEREQ exit).

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPSCOPE
Address of the 4-byte ENQSCOPE name to be used.

Return codes

UERCBYP
Bypass this request.

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

UERCSCPE
An ENQSCOPE name has been supplied.

XPI calls
All can be used.

API and SPI commands
All can be used, except for:
EXEC CICS SHUTDOWN
EXEC CICS XCTL

Note: Take care when issuing recursive commands. For example, you must avoid entering a loop when issuing an enqueue request from the XNQEREQ exit. Use of the recursion counter UEPRECUR is recommended.

Exit XNQEREQC

Exit XNQEREQC is invoked after an enqueue API request has completed, before return from the enqueue EXEC interface program.

Exit-specific parameters

UEPCLPS
Address of a copy of the command parameter list. See "The command-level parameter structure" on page 78.

UEPNQTOK
Address of a 4-byte area which can be used to pass information between XNQEREQ and XNQEREQC for a single enqueue request.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code EIBRCODE. For details of EIB return codes, see EIB fields, in the *CICS Application Programming Reference* manual.

UEPRESP
Address of a 4-byte binary copy of the EIB response code EIBRESP.

UEPRESP2
Address of a 4-byte binary copy of the EIB response code EIBRESP2.

UEPTSTOK
Address of a 4-byte token which can be used to pass information between successive enqueue requests within the same task (for example, between successive invocations of the XNQEREQC exit).

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPSCOPE
Address of the 4-byte ENQSCOPE name used.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

API and SPI commands
All can be used, except for:
EXEC CICS SHUTDOWN
EXEC CICS XCTL

You can update the copies of EIBRCODE, EIBRESP, and EIBRESP2 that you are given in the parameter list. If you update the values, CICS copies the new values into the application program's EIB after the completion of XNQEREQC or if you specify a return code of UERCBYP in XNQEREQ.

You must set valid enqueue responses. You must set all three of EIBRCODE, EIBRESP, and EIBRESP2 to a consistent set of values, such as would be set by the enqueue domain to describe a valid completion. CICS does not check the consistency of EIBRCODE, EIBRESP, and EIBRESP2. If EIBRCODE is set to a non-zero value and EIBRESP is set to zero, CICS will override EIBRESP with a non-zero value. To help you set values for EIBRCODE, EIBRESP, and EIBRESP2, the values used by the enqueue domain are specified in DSECT DFHNQUED.

Note: Take care when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when issuing an enqueue request from the XNQEREQC exit. Use of the recursion counter UEPRECUR is recommended.

The command-level parameter structure

The command-level parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of a bit string that describes the type of request and identifies each keyword specified with the request. The remaining addresses point to pieces of data associated with the request.

End of parameter list indicator

You can examine the EID to determine the type of request and the keywords specified. You can examine the other parameters in the list to determine the values of the keywords. You can also modify values of keywords specified on the request.

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you

modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

The UEPCLPS exit-specific parameter:

The UEPCLPS exit-specific parameter is included in both exit XNQEREQ and exit XNQEREQC, and contains the address of the command-level parameter structure.

The command-level parameter structure contains four addresses, NQ_ADDR0 through NQ_ADDR3. It is defined in the DSECT NQ_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHNQUED.

The command-level parameter list is made up as follows.

NQ_ADDR0
is the address of a 9-byte area called the EID, which is made up as follows:

- **NQ_GROUP**
- **NQ_FUNCT**
- **NQ_BITS1**
- **NQ_BITS2**
- **NQ_EIDOPT5**
- **NQ_EIDOPT6**
- **NQ_EIDOPT7**
- **NQ_EIDOPT8**

NQ_GROUP
Always X'12', indicating that this is a task control request.

NQ_FUNCT
One byte that defines the type of request:

X'04' ENQ

X'06' DEQ

NQ_BITS1
Existence bits that define which arguments were specified. To obtain the argument associated with a keyword, you need to use the appropriate address from the command-level parameter structure. Before using this address, you must check the associated existence bit. If the existence bit is set off, the argument was not specified in the request and the address should not be used.

X'80' Set if the request contains an argument for the RESOURCE keyword. If set, **NQ_ADDR1** is meaningful.

X'40' Set if the request contains an argument for the LENGTH keyword. If set, **NQ_ADDR2** is meaningful.

X'20' Set if the request contains an argument for the MAXLIFETIME keyword. If set, **NQ_ADDR3** is meaningful.

NQ_BITS2
Two bytes not used by the enqueue domain.

NQ_EIDOPT5
One byte not used by the enqueue domain.

NQ_EIDOPT6
One byte not used by the enqueue domain.

NQ_EIDOPT7
One byte not used by the enqueue domain.

NQ_EIDOPT8
Indicates whether certain keywords were specified on the request.

X'04' NOSUSPEND was specified.

X'02' DEQ was specified.

X'01' ENQ was specified.

NQ_ADDR1
is the address of an area containing the value from RESOURCE.

NQ_ADDR2
is the address of the halfword value of LENGTH.

NQ_ADDR3
is the address of the fullword value of MAXLIFETIME.

The relationship between arguments, keywords, data types, and input/output types is summarized for the enqueue commands in Table 4 on page 81.

Modifying fields in the command-level parameter structure:

The fields that are passed to the enqueue domain are used as input to the request. The correct method of modifying an input field is to create a new copy of it, and to change the address in the command-level parameter list to point to your new data.

Important:

1. Do not modify an input field by altering the data that is pointed to by the command-level parameter list. To do so would corrupt storage belonging to the application program and would cause a failure when the program attempted to reuse the field.
2. There are no output fields on EXEC CICS ENQ and DEQ requests.

Modifying the EID:

It is not possible to modify the EID to make major changes to requests. It is not possible, for example, to change an ENQ request to a DEQ request. However, you can make minor changes to requests, such as to turn on the existence bit for LENGTH.

The list that follows shows the bits in the EID that can be modified. Any attempt to modify any other part of the EID is ignored.

NQ_BITS1

X'40' The existence bit for LENGTH

X'20' The existence bit for MAXLIFETIME.

NQ_EIDOPT7
A user exit program at XNQEREQ can set the following on or off for ENQ commands:

X'04' The existence bit for NOSUSPEND.

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the enqueue request only.

Note: Your user exit program is prevented from making major changes to the EID. However, you must take great care when making the minor modifications that **are** permitted.

Use of the task token UEPTSTOK:

The task token, UEPTSTOK, provides the address of a 4-byte area that you can use to pass information between successive enqueue requests in the same task. For example, you can use the task token if you need to pass information between successive invocations of the XNQEREQ exit.

By contrast, UEPNQTOK is usable only for the duration of a single enqueue request, because its contents can be destroyed at the end of the request.

Note:

1. The lifetime of the area pointed to by UEPTSTOK is the lifetime of the task.
2. The value of UEPTSTOK is shared by all the exits to which it is passed during the lifetime of the task.

Table 4. User arguments and associated keywords, data types, and input/output types

Argument	Keyword	Data type	Input/output type
Arg1	RESOURCE	DATA-AREA	input
Arg2	LENGTH	BIN(15)	input
Arg3	MAXLIFETIME	CVDA	input

Modifying user arguments:

User exit programs can modify user input arguments either by obtaining/setting storage, or by setting a pointer.

User exit programs can modify user input arguments by:

1. Obtaining sufficient storage to hold the modified argument
2. Setting the storage to the required value
3. Setting the associated pointer in the parameter list to the address of the newly-acquired area.

Note:

1. CICS does not check changes to argument values, so any changes must be verified by the user exit program making the changes.
2. It is not advisable for XNQEREQC to modify input arguments.

Adding user arguments:

Global user exit programs can add arguments associated with the LENGTH and MAXLIFETIME keywords. You must ensure that the arguments you specify or modify in your exit programs are valid.

The valid values for MAXLIFETIME are DFHVALUE(TASK) and DFHVALUE(UOW), which are 233 and 246 respectively.

Assuming that the argument to be added does not already exist, the user exit program must:

1. Obtain storage for the argument to be added
2. Initialize the storage to the required value
3. Select and set up the appropriate pointer from the parameter list
4. Select and set up the appropriate argument existence bit in the EID
5. Modify the parameter list to reflect the new end of list indicator.

Removing user arguments:

User exit programs can remove arguments (for which the program is totally responsible) associated with the LENGTH and MAXLIFETIME keywords.

Assuming that the argument to be removed exists, the user exit program must:

1. Switch the corresponding argument existence bit to '0'b in the EID
2. Modify the parameter list to reflect the new end of list indicator.

Sample exit program, DFH$XNQE

CICS supplies a sample exit program, DFH$XNQE, for the XNQEREQ exit.

The program gives examples of:

- Coding Exec Interface Global User Exits
- Issuing a mixture of XPI and EXEC CICS API calls within Global User Exits
- Three methods of adding a SCOPE value to exec ENQ and DEQ requests, so that they apply to multiple regions within the Sysplex. Methods A and B force a match to an installed ENQMODEL resource definition. Method C bypasses the use of ENQMODEL resource definitions even if there would have been a match.

  The methods are:

  Method A

  Prefix the Resource name with a 1- to 255-character value (this sample uses a 4-character value) for the ENQNAME on the ENQMODEL resource definition to which you want to force a match. The exit terminates and processing continues as though the chosen ENQMODEL had been matched normally. The scope is then supplied by the matched ENQMODEL definition.

  This method applies only to resource names shorter than 255-n (where n is the length of you chosen prefix).

  Method B

  Similar to method A, but you replace the first 1- to 8-characters of the resource name with your chosen string instead of prefixing it. This method:

  – applies only to resource names of length equal to or greater than that of your replacement string.
  – is an alternative to method A when a resource name too long to allow the use of that method.

  Method C

  Place a 4-character Scope value in UEPSCOPE, and return UERCSCPE in R15. This will bypass any installed ENQMODEL resource definition, forcing a Sysplex Scope ENQ/DEQ request.

This method is not recommended if you have an ENQMODEL table, because the latter is designed to preserve data integrity by preventing the possibility of a region scope enqueue and a sysplex scope enqueue (or two sysplex scope enqueues with different scopes) existing for the same resource. (Because sysplex and region scope enqueues use separate namespaces, a region scope enqueue will never wait on a sysplex enqueue, nor will a sysplex scope enqueue wait on a region enqueue.)

Related concepts:

"Enqueue EXEC interface sample exit program (DFH$XNQE)" on page 24
CICS includes a sample global user exit program for the enqueue EXEC interface. The sample program is DFH$XNQE.

Using XNQEREQ to alter the ENQ or DEQ scope:

There are three points to consider when modifying the scope of ENQ or DEQ.

1. XNQEREQ enables you to allow existing applications to be converted to use sysplex enqueues without changing the application.

   Note: Use of either the ENQMODEL resource definition or the user exit allows this in most cases, but those applications where the resource name is determined dynamically and not known in advance can only be so converted by use of this exit.
2. Sysplex and region scope enqueues use separate namespaces. A region scope enqueue will never wait on a sysplex enqueue, nor will a sysplex scope enqueue wait on a region enqueue.

   Note: This situation can only arise when you use the exit. Use of the ENQMODEL resource definitions as your only method of defining the SCOPE of an ENQ or DEQ avoids this potential risk.
3. Both region and sysplex scope are supported for string ENQs, but sysplex scope is not supported for address ENQ.

Event capture exit XEPCAP

The XEPCAP exit is invoked just before an event is captured by CICS event processing. Use the XEPCAP exit to detect when events are captured.

Exit-specific parameters

UEPEPCX
Address of the EPCX (event context data structure). This parameter contains information about the event being captured. For more information about EPCX, see EPCX Event Processing Context Container.

UEPEPTASK
Address of a 4-byte (packed decimal) field containing the task number.

UEPLOAD
Address of the capturing transaction program load point.

UEPRSA
Address of the capturing transaction register save area, which contains the contents of the registers at the point when the program issued the EXEC CICS command. If the event is captured from program initialization rather than an API call then this parameter is set to 0.

Return codes

UERCNORM
Continue processing.

XPI calls
You can use all XPI calls.

API and SPI commands
No EXEC CICS commands can be used.

EXEC interface program exits XEIIN, XEIOUT, XEISPIN, and XEISPOUT

There are four global user exit points in the EXEC interface program that you can use before or after an API or SPI call.

XEIIN Invoked before the execution of any EXEC CICS application programming interface (API) or system programming interface (SPI) command.

XEISPIN
Invoked before the execution of any EXEC CICS SPI command except:

- **EXEC CICS ENABLE**
- **EXEC CICS DISABLE**
- **EXEC CICS EXTRACT EXIT**
- **EXEC CICS PERFORM DUMP**
- **EXEC CICS RESYNC ENTRYNAME**

The sequence is:

```
TRACE  –  XEIIN  –  XEISPIN  –  EDF  –  command
```

XEIOUT
Invoked after the execution of any EXEC CICS API or SPI command.

XEISPOUT
Invoked after the execution of any EXEC CICS SPI command except those listed for XEISPIN.

The sequence is:

```
command  –  EDF  –  XEISPOUT  –  XEIOUT  –  TRACE
```

Note: Asynchronous processing of these exits might occur if the transaction is suspended; for example, during file I/O wait. This situation might also occur under CEDF because CEDF issues its own EXEC CICS commands between the application's XEISPIN and XEISPOUT exits.

If, for example, the same GWA is shared between the XEIIN and XEIOUT exits, you must allow for the possibility of asynchronous processing, in order to ensure integrity of the data and to prevent unpredictable results.

On entry to the exits, the exit-specific parameter UEPARG contains the address of the command parameter list.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The command parameter list

The first parameter in the list points to a string of data known as *argument 0*. The other parameters point to the values specified for the parameters passed on the command.

Argument 0 begins with a 2-byte function code that identifies the command. Function codes are documented in EIB fields in Reference -> Application development and in EXEC interface block (EIB) response and function codes in Reference > System programming. The function code is followed by a 2-byte field that contain "existence bits", which indicate whether arguments are passed on the command. For example, consider the command:

```
EXEC CICS LINK PROGRAM('MYPROG')
```

Here, argument 0 begins with the function code X'0E02' (LINK). Existence bit 1 is set, indicating that there is an argument 1 (namely, 'MYPROG').

The correspondence between command parameters (such as PROGRAM) and their positions and values in the parameter list (in this case, argument 1, 'MYPROG') can be deduced from the translated code for the particular command.

Important:

Modifying CICS commands by changing argument 0 is not supported, and leads to unexpected errors or results.

For example, if an application program is written in assembler or PL/I and you modify argument 0, you will be writing to program storage (that is, storage occupied by the program itself), which could cause 0C4 abends. Furthermore, modifying argument 0 not only alters the CICS command for this execution of the command in the application program, it changes the CICS command in the virtual storage copy of the application program. This means that the next task to invoke the same copy of the program will also execute the modified command.

This particular example of the danger of modifying argument 0 does not apply to COBOL or C application programs, but nevertheless you should not modify CICS commands for application programs written in any supported language.

Bypassing commands

An XEIIN or XEISPIN exit program can bypass execution of a command by setting the UERCBYP return code. If it does this, EDF is not invoked, but XEISPOUT, XEIOUT, and exit trace are invoked if they are active.

Bypassing an EXEC CICS command allows an exit program to replace the CICS function with its own processing, for example.

Before setting UERCBYP, your program should check the value pointed to by UEPPGM, to ensure that it is not bypassing an EXEC CICS command issued by CICS.

Exit XEIIN

Exit XEIIN is invoked before the execution of any EXEC CICS API or SPI command.

Exit-specific parameters

UEPARG
Address of the EXEC command parameter list.

UEPEXECB
Address of the system EIB.

UEPUSID
Address of the 8-character userid.

UEPPGM
Address of the 8-character application program name.

UEPLOAD
Address of the application program's load-point.

UEPRSA
Address of the application's register save area. This contains the contents of the registers at the point when the program issued the EXEC CICS command.

UEP_EI_PBTOK
Address of a 4-byte field containing the z/OS Workload Manager (WLM) Performance Block Token. An exit program can use this token to access information (such as the service class token, SERVCLS) in the WLM Performance Block. To do so, it must use the WLM EXTRACT macro, IWMMEXTR, passing the Performance Block Token as the MONTKN input parameter. For more information about the IWMMEXTR macro, see *z/OS MVS Programming: Workload Management Services*.

An exit program must not attempt to modify the Performance Block: if it does so, the results are unpredictable.

Return codes

UERCNORM
Continue processing.

UERCBYP
Bypass the execution of this command.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Exit XEISPIN

Exit XEISPIN is invoked before the execution of some SPI commands. The exit is not invoked for the **ENABLE**, **DISABLE**, **EXTRACT EXIT**, **PERFORM DUMP** and **RESYNC ENTRYNAME** commands.

Exit-specific parameters

UEPARG
Address of the EXEC command parameter list.

UEPEXECB
Address of the system EIB.

UEPUSID
Address of the 8-character userid.

UEPPGM
Address of the 8-character application program name.

UEPLOAD
Address of the load-point of the application program.

UEPRSA
Address of the register save area of the application program. This area contains the contents of the registers at the point when the program issued the EXEC CICS command.

UEP_EI_PBTOK
Address of a 4-byte field that contains the z/OS Workload Manager (WLM) Performance Block Token. An exit program can use this token to access information in the WLM Performance Block, for example, the service class token, SERVCLS. To do so, it must use the WLM EXTRACT macro, IWMMEXTR, passing the Performance Block Token as the MONTKN input parameter. For more information about the IWMMEXTR macro, see z/OS MVS Programming: Workload Management Services.

An exit program must not attempt to modify the Performance Block: if it does so, the results are unpredictable.

Return codes

UERCNORM
Continue processing.

UERCBYP
Bypass the execution of this command.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Exit XEIOUT

Exit XEIOUT is invoked after the execution of any EXEC CICS API or SPI command.

Exit-specific parameters

UEPARG
Address of the EXEC command parameter list.

UEPEXECB
Address of the system EIB.

UEPUSID
Address of the 8-character userid.

UEPPGM
Address of the 8-character application program name.

UEPLOAD
Address of the application program's load-point.

UEPRSA
Address of the application's register save area. This contains the contents of the registers at the point when the program issued the EXEC CICS command.

UEP_EI_PBTOK
Address of a 4-byte field containing the z/OS Workload Manager (WLM) Performance Block Token. An exit program can use this token to access information (such as the service class token, SERVCLS) in the WLM Performance Block. To do so, it must use the WLM EXTRACT macro, IWMMEXTR, passing the Performance Block Token as the MONTKN input parameter. For more information about the IWMMEXTR macro, see *z/OS MVS Programming: Workload Management Services*.

An exit program must not attempt to modify the Performance Block: if it does so, the results are unpredictable.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Exit XEISPOUT

Exit XEISPOUT is invoked after the execution of some SPI commands. The exit is not invoked for the **ENABLE**, **DISABLE**, **EXTRACT EXIT**, **PERFORM DUMP** and **RESYNC ENTRYNAME** commands.

Exit-specific parameters

UEPARG
Address of the EXEC command parameter list.

UEPEXECB
Address of the system EIB.

UEPUSID
Address of the 8-character userid.

UEPPGM
Address of the 8-character application program name.

UEPLOAD
Address of the application program's load-point.

UEPRSA
Address of the application's register save area. This contains the contents of the registers at the point when the program issued the EXEC CICS command.

UEP_EI_PBTOK
Address of a 4-byte field containing the z/OS Workload Manager (WLM) Performance Block Token. An exit program can use this token to access information (such as the service class token, SERVCLS) in the WLM Performance Block. To do so, it must use

the WLM EXTRACT macro, IWMMEXTR, passing the Performance Block Token as the MONTKN input parameter. For more information about the IWMMEXTR macro, see *z/OS MVS Programming: Workload Management Services*.

An exit program must not attempt to modify the Performance Block: if it does so, the results are unpredictable.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Front End Programming Interface exits XSZARQ and XSZBRQ

If you have installed the Front End Programming Interface (FEPI), you can use global user exits XSZARQ and XSZBRQ before and after FEPI commands.

XSZBRQ
Invoked before a FEPI command is executed (but after the syntax of the command has been validated, and therefore after EDF processing).

XSZARQ
Invoked immediately after a FEPI command has completed (before EDF processing).

Note that both the FEPI application programming and system programming commands cause XSZBRQ and XSZARQ to be invoked, but the latter do not provide the exit programs with any meaningful information.

You cannot use exit programming interface (XPI) calls or EXEC CICS commands in programs invoked from these exits. The exits allow you to monitor the FEPI commands and data being processed; you can inhibit commands, and modify specific command options. You could use them for:

- Monitoring the issue of FEPI commands
- Workload routing
- External security on application programming commands.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

XSZBRQ

XSZBRQ is invoked before a FEPI command is executed; the input parameters for the command are passed to the exit program.

The majority of the information passed is read-only, but you can write a program to update specific parameters. FEPI does not check the validity of the new values for the updated parameters. In addition, your exit program can decide whether the request is to be processed or bypassed. You could use XSZBRQ, for example, to log commands, to bypass commands that violate the conventions of your installation, or to reroute commands by changing their specified targets or pools.

Together, UEPSZALP and UEPSZALT contain the information necessary to initiate a conversation.

When invoked

Invoked by FEPI before a FEPI command runs, but after syntax and semantic checking.

Exit-specific parameters

UEPSZACT

A 2-byte field that identifies the command. The values are given in Table 5 on page 92.

UEPSZCNV

An 8-character field containing the conversation ID (CONVID) for the command. Applicable on FEPI ALLOCATE, SEND, RECEIVE, CONVERSE, EXTRACT, ISSUE, START, and FREE commands.

UEPSZALP

An 8-character field containing the name of the pool (POOL). Modifiable and applicable on FEPI ALLOCATE and CONVERSE commands.

UEPSZALT

An 8-character field containing the name of the target (TARGET). Modifiable and applicable on FEPI ALLOCATE and CONVERSE commands.

UEPSZTIM

Fullword binary field containing the time-out value (TIMEOUT). Modifiable and applicable on FEPI ALLOCATE, RECEIVE, CONVERSE, and START commands.

UEPSZSND

Address of the 'send' data-area (FROM). Applicable on FEPI CONVERSE and SEND commands.

UEPSZSNL

Fullword binary field containing the length of the 'send' data (FROMFLENGTH, FLENGTH). Applicable on FEPI CONVERSE and SEND commands.

UEPSZSTT

A 4-character field containing the transaction ID (TRANSID). Modifiable and applicable on FEPI START commands.

UEPSZSTM

A 4-character field containing the terminal ID (TERMID). Modifiable and applicable on FEPI START commands.

UEPSZSNK

A 1-bit flag field indicating whether data is in key stroke format (KEYSTROKE). Applicable on FEPI CONVERSE FORMATTED and SEND FORMATTED commands. It can contain the following values:

UEPSZSNK_OFF

Not key stroke format.

UEPSZSNK_ON

Key stroke format.

UEPSZSNE
A 1-character field containing the key stroke escape character (ESCAPE). Applicable on FEPI CONVERSE FORMATTED and SEND FORMATTED commands.

Return codes

UERCNORM
Continue processing.

UERCBYP
Do not process the request; return INVREQ to the application.

Note: Your exit program cannot bypass events (like CICS shutdown or end-of-task).

XPI calls
Do not use XPI calls.

XSZARQ

XSZARQ is invoked immediately after a FEPI command has been executed; the exit program is passed the parameters that are output from the command. All of the information passed is read-only.

When invoked
Invoked by FEPI immediately after a FEPI command has been processed.

Exit-specific parameters

UEPSZACN
A 2-byte field that identifies the command. The values are given in Table 5 on page 92.

UEPSZCON
An 8-character field containing the conversation ID (CONVID) for the command. Applicable on FEPI ALLOCATE, SEND, RECEIVE, CONVERSE, EXTRACT, ISSUE, START, and FREE commands.

UEPSZRP2
Fullword containing the response code for the command (RESP2).

UEPSZRVD
Address of the 'receive' data-area (INTO). Applicable on FEPI RECEIVE, CONVERSE, and EXTRACT FIELD commands.

UEPSZRVL
Fullword binary data field containing the length of the receive data (FLENGTH, TOFLENGTH). Applicable on FEPI RECEIVE, CONVERSE, and EXTRACT FIELD commands.

Return code

UERCNORM
Continue processing.

XPI calls
Do not use any XPI calls.

The UEPSZACT and UEPSZACN exit-specific parameters

Both XSZBRQ and XSZARQ are passed a parameter (**UEPSZACT** for XSZBRQ, and **UEPSZACN** for XSZARQ) indicating the command or event being processed.

Table 5. relates the hexadecimal values passed in UEPSZACT and UEPSZACN to the FEPI commands they represent.

Table 5. Settings of UEPSZACT for exit XSZBRQ and UEPSZACN for exit XSZARQ

Name	Setting (hex)	FEPI command or event
UEPSZNOA	820E	AP NOOP
UEPSZOAL	8210	ALLOCATE
UEPSZOCF	8212	CONVERSE FORMATTED
UEPSZOCD	8214	CONVERSE DATASTREAM
UEPSZOXC	8216	EXTRACT CONV
UEPSZOXF	8218	EXTRACT FIELD
UEPSZOXS	821A	EXTRACT STSN
UEPSZOFR	821C	FREE
UEPSZOSU	821E	ISSUE
UEPSZORF	8220	RECEIVE FORMATTED
UEPSZORD	8222	RECEIVE DATASTREAM
UEPSZOSF	8224	SEND FORMATTED
UEPSZOSD	8226	SEND DATASTREAM
UEPSZOST	8228	START
UEPSZSDN	8402	CICS normal shutdown 1
UEPSZSDI	8404	CICS immediate shutdown 1
UEPSZSDF	8406	CICS forced shutdown 1
UEPSZEOT	8408	CICS end-of-task 1
UEPSZNOS	840E	SP NOOP
UEPSZOQY	8422	INQUIRE PROPERTYSET
UEPSZOIY	8428	INSTALL PROPERTYSET
UEPSZODY	8430	DISCARD PROPERTYSET
UEPSZOQN	8442	INQUIRE NODE
UEPSZOTN	8444	SET NODE
UEPSZOIN	8448	INSTALL NODELIST
UEPSZOAD	844A	ADD POOL
UEPSZODE	844C	DELETE POOL
UEPSZODN	8450	DISCARD NODELIST
UEPSZOQP	8462	INQUIRE POOL
UEPSZOTP	8464	SET POOL
UEPSZOIP	8468	INSTALL POOL
UEPSZODP	8470	DISCARD POOL
UEPSZOQT	8482	INQUIRE TARGET
UEPSZOTT	8484	SET TARGET
UEPSZOIT	8488	INSTALL TARGETLIST
UEPSZODT	8490	DISCARD TARGETLIST
UEPSZOQC	84A2	INQUIRE CONNECTION
UEPSZOTC	84A4	SET CONNECTION

Note:

- 1 These events are generated internally by CICS; you cannot bypass them.

Using XMEOUT to control message output

You can use the XMEOUT global user exit, in the CICS message domain, to suppress or reroute FEPI messages.

Note, however, that error conditions that generate a message also generate a transient data queue record. It is more efficient to handle such events using a monitoring program, through the TD queue, than by duplicating a message and then acting on it.

File control domain exits, XFCFRIN and XFCFROUT

The XFCFRIN exit is invoked on entry to the main file control request gate, FCFR, and the XFCFROUT exit runs after the completion of a file control request. The XFCFRIN and XFCFROUT exits must be coded to threadsafe standards and declared threadsafe to take advantage of threadsafe remote file support.

XFCFRIN

XFCFRIN allows you to write a program to perform one or more of the following tasks:

- Monitor file control requests and allow them to continue, to be processed by CICS file control
- Intercept file control requests and bypass CICS file control processing altogether
- Redirect the request to a remote region.

If the exit program passes the request to CICS file control (without choosing to redirect it to a remote region), it is not allowed to make changes to any of the parameters. If the exit program intercepts the request and bypasses file control:

- It must return all the responses and output parameters that would otherwise have been returned by file control. These are marked **output** in the descriptions of the exit-specific parameters.
- It must indicate whether, if the request was function-shipped, the mirror transaction is permitted to terminate. Certain file control requests require that another request has been executed previously in the same transaction. (For example, READNEXT must be preceded by a matching STARTBR; REWRITE must be preceded by a matching READ, READNEXT, or READPREV with the UPDATE option). If the mirror transaction terminates between two such requests, the second is likely to fail. Conversely, a mirror transaction that is retained unnecessarily will hold on to CICS resources and may contribute to storage and locking problems.
- CICS terminates file browses and outstanding updates as part of syncpoint processing. However, the XFCFRIN exit is not invoked for syncpoint. If you want to emulate this aspect of CICS behavior accurately, or you want to support recoverable resources, you must invoke a task-related user exit program which schedules the syncpoint manager—see “Coding a program to be started by the CICS sync point manager” on page 366.

To redirect the request to a remote region, the exit program must add or change the value of the SYSID parameter. In this case, it may also need to supply the values of the key length and record length. It is not permitted to make changes to any of the other parameters.

XFCFROUT

XFCFROUT allows you to monitor the results of completed file control requests. For example, if you didn't choose to bypass CICS file control processing, you can analyze the (CICS-internal) file control request to determine its type, the parameters passed to file control, and the values returned. It is invoked in both the following cases:

- After CICS file control has completed its processing, either normally or with an error
- If your XFCFRIN exit program chooses to bypass CICS file control processing.

All parameters are input-only; your exit program cannot modify any of the values.

To use IPIC connections for function shipping file control requests, ensure that XFCFRIN and XFCFROUT check that the UEPTERM parameter is a non-zero value before trying to use it as an address. The UEPTERM parameter is a zero for file control requests that have been function shipped over an IPIC connection.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XFCFRIN

Exit XFCFRIN is invoked before the execution of a file control request.

The request can have originated from:

- The execution of an application request to process a user file
- The receipt of a function-shipped request
- An internal CICS request to process a system file.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Zero, or the address of the 4-byte terminal ID. If no address is returned, this could mean that this request has been function shipped over an IPIC connection.

UEPPROG
Address of the 8-byte application program name.

UEP_FC_FUNCTION
Address of a byte containing the function. The FCFR functions are derived from those available through the EXEC CICS interface, where certain request options (SET, INTO, UPDATE) have been included in the function values. For example, UEP_FC_FUN_DELETE is derived from EXEC CICS DELETE with

the RIDFLD option specified; UEP_FC_FUN_REWRITE_DELETE is derived from EXEC CICS DELETE without RIDFLD. The possible values are:

- UEP_FC_FUN_READ_INTO
- UEP_FC_FUN_READ_SET
- UEP_FC_FUN_READ_UPDATE_INTO
- UEP_FC_FUN_READ_UPDATE_SET
- UEP_FC_FUN_WRITE
- UEP_FC_FUN_REWRITE
- UEP_FC_FUN_REWRITE_DELETE
- UEP_FC_FUN_DELETE
- UEP_FC_FUN_UNLOCK
- UEP_FC_FUN_START_BROWSE
- UEP_FC_FUN_READ_NEXT_INTO
- UEP_FC_FUN_READ_NEXT_SET
- UEP_FC_FUN_READ_PREVIOUS_INTO
- UEP_FC_FUN_READ_PREVIOUS_SET
- UEP_FC_FUN_READ_NEXT_UPDATE_INTO
- UEP_FC_FUN_READ_NEXT_UPDATE_SET
- UEP_FC_FUN_READ_PREVIOUS_UPDATE_INTO
- UEP_FC_FUN_READ_PREVIOUS_UPDATE_SET
- UEP_FC_FUN_RESET_BROWSE
- UEP_FC_FUN_END_BROWSE

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEP_FC_FILE_NAME
Address of an 8-byte modifiable field containing the filename.

UEP_FC_BUFFER_P
Address of a fullword containing the address of the buffer provided by the originator of the request, in which the (**output**) record is to be returned on completion of a READ, READ NEXT, or READ PREV request with the INTO option.

UEP_FC_BUFFER_L
Address of a fullword containing (for READ, READ NEXT, and READ PREV requests) the value of the LENGTH of the buffer into which the record is to be read.

UEP_FC_RECORD_P
Address of one of the following:

- If the request is a READ, READ NEXT, or READ PREV with the SET option, a fullword in which is to be returned the address (**output**) of a buffer, into which the record will be placed. The buffer itself is supplied either by CICS file control or, if the exit program bypasses file control, by the exit program.
- If the request is WRITE or REWRITE, a fullword containing the address of the record to be written.

UEP_FC_RECORD_L
Address of a fullword containing (for READ, WRITE, REWRITE, READ NEXT, and READ PREV requests) the value of LENGTH.

For all READ, READ NEXT, or READ PREV requests, this is an **output** field, in which the actual length of the record read is placed on return.

Warning: For requests that specify INTO, do not change the value of LENGTH to a value greater than the value specified by the UEP_FC_BUFFER_L field. To do so could cause a storage overlay in the application.

For a WRITE or REWRITE, this is an optional field which, if present, contains the length of the record to be written. If the field is not specified, the fullword contains binary zeroes. In this case, to modify the record length for a remote file use UEP_FC_M_RECORD_L instead.

See also the description of the UEP_FC_SYSID parameter.

UEP_FC_MAX_RECORD_L
Address of a fullword containing the (**output**) maximum record length of the file. (CICS function shipping uses this value to update the file's entry in the remote file control table.)

UEP_FC_RECORD_ID_P
Address of a fullword containing the address of the RIDFLD (record identifier) value. For a discussion of when the record identifier is an input or an **output** field, see Table 6 on page 112.

UEP_FC_RECORD_ID_L
Address of the halfword value of KEYLENGTH, which is the (possibly partial) length of the record identifier.

KEYLENGTH is an optional input parameter on READ, WRITE, DELETE, START BR, READ NEXT, READ PREV, and RESET BR requests. If the field is not specified, the halfword contains binary zeroes. In this case, to modify the key length for a remote file use UEP_FC_M_RECORD_ID_L instead.

See also the description of the UEP_FC_SYSID parameter.

UEP_FC_FULL_RECORD_ID_L
Address of the halfword value of the full length of the record identifier.

The full length of the record identifier is returned as a mandatory **output** field on READ NEXT and READ PREV requests. The value is used by CICS function shipping.

UEP_FC_RECORD_ID_TYPE
Address of a byte containing (for READ, WRITE, DELETE, START BR, READ NEXT, READ PREV, and RESET BR requests) the RIDFLD type. On input to the exit, this parameter will be set to one of:

UEP_FC_KEY
VSAM KSDS or AIX® PATH access

UEP_FC_RBA
VSAM ESDS or KSDS via RBA access

UEP_FC_RRN
VSAM RRDS access

UEP_FC_DEBKEY
BDAM deblocking by key (READ, DELETE, START BR, and RESET BR requests only)

UEP_FC_DEBREC
BDAM deblocking by relative record (READ, DELETE, START BR, and RESET BR requests only)

UEP_FC_XRBA
VSAM extended ESDS access

UEP_FC_REQID
Address (for START BR, READ NEXT, READ PREV, RESET BR, and END BR requests) of the halfword value of REQID.

UEP_FC_NUMREC
Address of the fullword value of NUMREC (**output**), in which (if the request is DELETE with RIDFLD) the number of records that have been deleted is returned.

UEP_FC_KEY_COMPARISON
Address of a byte containing (for READ, START BR, and RESET BR requests) the key comparison setting. On input to the exit, this parameter will be set to one of:

UEP_FC_EQUAL
Key-equal-to comparison is to be used.

UEP_FC_GTEQ
Key-greater-than-or-equal-to comparison is to be used.

UEP_FC_GENERIC
Address of a byte containing (for READ, DELETE, START BR, and RESET BR requests) the generic key setting. On input to the exit, this parameter will be set to one of:

UEP_FC_GENERIC_KEY
Generic key is to be used for key search.

UEP_FC_FULL_KEY
Full key is to be used for key search.

UEP_FC_MASS_INSERT
Address of a byte containing (for WRITE requests) the mass insert setting. On input to the exit, this parameter will be set to one of:

UEP_FC_SEQUENTIAL_WRITE
Records are to be written in sequential mode.

UEP_FC_DIRECT_WRITE
Records are to be written in direct mode.

UEP_FC_READ_INTEGRITY
Address of a byte containing (for non-update READ, READ NEXT, and READ PREV requests) the read integrity setting. (In current versions of CICS, this setting applies only to VSAM RLS.) On input to the exit, this parameter will be set to one of:

UEP_FC_CR
Consistent read integrity is to be used.

UEP_FC_FCT_VALUE
Read integrity is according to the setting in the FILE definition.

UEP_FC_NRI
The record is to be read with no read integrity.

UEP_FC_RR
Repeatable read integrity is to be used.

UEP_FC_TOKEN
Address of a fullword containing the value of TOKEN.

If the request is READ, READ NEXT, or READ PREV with update, and the address is not null, the area is an **output** field in which the token is to be returned.

If the request is REWRITE, DELETE without RIDFLD, or UNLOCK, the area is an input field.

UEP_FC_SYSID
Address of a 4-byte area that is to contain the SYSID identifying the remote region. On input to the exit, the area contains either:
- The value of the SYSID option of the API call, *or*
- Blanks (if SYSID was not specified).

To redirect the request to a different region, the exit program must place the SYSID of the target region in this **output** area.

If this parameter is set by the exit program, the request is function-shipped by file control without any reference to the file's attributes. If the key length has not been included on the request, the exit program must establish its value by setting the UEP_FC_RECORD_ID_L parameter.

Similarly, if the request is WRITE or REWRITE, and the record length has not been specified on the request, the exit program must establish its value by setting the UEP_FC_RECORD_L parameter.

UEP_FC_LENGTH_ERROR_CODE
Address of a 1-byte **output** area containing the length error code to be returned after a request has completed. The possible values are:
- UEP_FC_LENGTH_OK
- UEP_FC_BUFFER_LEN_TOO_SMALL
- UEP_FC_RECORD_LEN_TOO_LARGE
- UEP_FC_BUFFER_LEN_NOT_FILE_LEN
- UEP_FC_RECORD_LEN_NOT_FILE_LEN

UEP_FC_DUPLICATE_KEY_CODE
Address of a 1-byte **output** area indicating whether the request found more than one record for the supplied key. The possible values are:
- UEP_FC_DUPLICATE KEY
- UEP_FC_NOT_DUPLICATE KEY

UEP_FC_ACCMETH_RETURN_CODE
Address of a 4-byte **output** area in which access-method-dependent information is to be returned when either of the responses UEP_FC_REASON_ACCMETH_REQUEST_ERROR or UEP_FC_REASON_IO_ERROR is returned.

The returned value is placed in bytes 2–5 of the EIBRCODE.

UEP_FC_RESPONSE
Address of a 1-byte **output** area containing the response after a request has completed:

UEP_FC_RESPONSE_OK
Processing has completed without errors.

UEP_FC_RESPONSE_EXCEPTION
Processing has completed with an error condition. The reason is set in UEP_FC_REASON.

UEP_FC_RESPONSE_DISASTER
An error has occurred which prevents processing from completing. Typically, this is as a result of a DISASTER response from an XPI function call, or corruption of data addressed from UEPGAA or UEPTSTOK.

If you set this response, the caller of file control will assume that first-failure data capture has been performed. If you are percolating a DISASTER response from an XPI request, first-failure data capture will have been performed already; if not, you should attempt to capture sufficient information to successfully diagnose the error. The DFHDUDUX SYSTEM_DUMP XPI function may be suitable for this purpose.

UEP_FC_RESPONSE_INVALID
The exit program was invoked with an invalid parameter list, indicating a CICS internal logic error. Note that an invalid parameter list that indicates an application error should give an EXCEPTION response.

UEP_FC_RESPONSE_PURGED
An XPI function call has received a PURGED response. Setting this response is equivalent to setting the UERCPURG return code, except that any changes to the parameter list are honored.

UEP_FC_REASON
Address of a 1-byte **output** area containing, after a request has completed with an EXCEPTION response, the reason. The possible reasons are:

- UEP_FC_REASON_ACCMETH_REQUEST_ERROR
- UEP_FC_REASON_DELETE_AFTER_READ_UPDATE
- UEP_FC_REASON_DELETE_BEFORE_READ_UPDATE
- UEP_FC_REASON_DUPLICATE_READ_UPDATE
- UEP_FC_REASON_DUPLICATE_RECORD
- UEP_FC_REASON_DUPLICATE_REQID
- UEP_FC_REASON_END_OF_FILE
- UEP_FC_REASON_FILE_DISABLED
- UEP_FC_REASON_FILE_NOT_OPEN
- UEP_FC_REASON_FILE_NOT_FOUND
- UEP_FC_REASON_FULL_KEY_WRONG_LENGTH
- UEP_FC_REASON_GENERIC_DELETE_NOT_KSDS
- UEP_FC_REASON_GENERIC_KEY_TOO_LONG
- UEP_FC_REASON_ILLEGAL_KEY_TYPE_CHANGE
- UEP_FC_REASON_INSUFFICIENT_SPACE
- UEP_FC_REASON_INVALID_UPDATE_TOKEN
- UEP_FC_REASON_IO_ERROR
- UEP_FC_REASON_KEY_LENGTH_NEGATIVE
- UEP_FC_REASON_KSDS_AND_XRBA
- UEP_FC_REASON_NO_VARIABLE_LENGTH
- UEP_FC_REASON_NOTAUTH
- UEP_FC_REASON_NOT_EXTENDED
- UEP_FC_REASON_READPREV_IN_GENERIC_BROWSE

- UEP_FC_REASON_RECORD_NOT_FOUND
- UEP_FC_REASON_REWRITE_BEFORE_READ_UPDATE
- UEP_FC_REASON_RIDFLD_KEY_NOT_RECORD_KEY
- UEP_FC_REASON_UNKNOWN_REQID_ENDBR
- UEP_FC_REASON_UNKNOWN_REQID_READNEXT
- UEP_FC_REASON_UNKNOWN_REQID_READPREV
- UEP_FC_REASON_UNKNOWN_REQID_RESETBR

UEP_FC_EXIT_TOKEN
Address of a 4-byte token to be passed to XFCFROUT. This allows you, for example, to pass a work area to exit XFCFROUT.

UEP_FC_M_RECORD_L
Address of a fullword containing the modified record length field. You can use this to change the LENGTH, for remote files only, when the LENGTH field is not specified on the API call. If the LENGTH field is specified on the API call, to change the LENGTH use UEP_FC_RECORD_L instead, because changing the value at the address stored by UEP_FC_M_RECORD_L has no effect.

UEP_FC_M_RECORD_ID_L
Address of a halfword containing the modified key length field. You can use this to change the KEYLENGTH, for remote files only, when the KEYLENGTH field is not specified on the API call. If the KEYLENGTH field is specified, to change the KEYLENGTH use UEP_FC_RECORD_ID_L instead, because changing the value at the address stored by UEP_FC_M_RECORD_ID_L has no effect.

Return codes

UERCNORM
Continue processing.

UERCBYP
Bypass CICS processing of this request. If the exit program has been invoked for a function-shipped request, the mirror transaction is permitted to terminate.

UERCBYPL
Bypass CICS processing of this request. If the exit program has been invoked for a function-shipped request, the mirror transaction must not terminate.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

API and SPI calls
None can be used.

Exit XFCFROUT

Exit XFCFROUT is invoked after the completion of a file control request.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Zero, or the address of the 4-byte terminal ID. If no address is returned, this could mean that this request has been function shipped over an IPIC connection.

UEPPROG
Address of the 8-byte application program name.

UEP_FC_FUNCTION
Address of a byte containing the function. The possible values are:
- UEP_FC_FUN_READ_INTO
- UEP_FC_FUN_READ_SET
- UEP_FC_FUN_READ_UPDATE_INTO
- UEP_FC_FUN_READ_UPDATE_SET
- UEP_FC_FUN_WRITE
- UEP_FC_FUN_REWRITE
- UEP_FC_FUN_REWRITE_DELETE
- UEP_FC_FUN_DELETE
- UEP_FC_FUN_UNLOCK
- UEP_FC_FUN_START_BROWSE
- UEP_FC_FUN_READ_NEXT_INTO
- UEP_FC_FUN_READ_NEXT_SET
- UEP_FC_FUN_READ_PREVIOUS_INTO
- UEP_FC_FUN_READ_PREVIOUS_SET
- UEP_FC_FUN_READ_NEXT_UPDATE_INTO
- UEP_FC_FUN_READ_NEXT_UPDATE_SET
- UEP_FC_FUN_READ_PREVIOUS_UPDATE_INTO
- UEP_FC_FUN_READ_PREVIOUS_UPDATE_SET
- UEP_FC_FUN_RESET_BROWSE
- UEP_FC_FUN_END_BROWSE

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEP_FC_FILE_NAME
Address of an 8-byte field containing the filename.

UEP_FC_BUFFER_P
Address of a fullword containing the address of the buffer provided by the originator of the request, in which the record is returned on completion of a READ, READ NEXT, or READ PREV request with the INTO option.

UEP_FC_BUFFER_L
Address of a fullword containing (for READ, READ NEXT, and READ PREV requests) the value of the LENGTH of the buffer into which the record was read.

UEP_FC_RECORD_P
Address of one of the following:
- If the request is a READ, READ NEXT, or READ PREV request with the SET option, a fullword in which is returned the address of a buffer, into which the record was placed.
- If the request is WRITE or REWRITE, a fullword containing the address of the record that was written.

UEP_FC_RECORD_L
Address of a fullword containing (for READ, WRITE, REWRITE, READ NEXT, and READ PREV requests) the value of LENGTH.

For all READ, READ NEXT, or READ PREV requests, this is an output field, containing the actual length of the record read. For these types of request, this record-length value is always present, even if the LENGTH option was not specified on the EXEC CICS API call.

For a WRITE or REWRITE, this is an optional field which, if present, contains the length of the record that was written.

UEP_FC_MAX_RECORD_L
Address of a fullword containing the maximum record length of the file.

UEP_FC_RECORD_ID_P
Address of a fullword containing the address of the value of RIDFLD (record identifier). For a discussion of when the record identifier is an input or an output field, see Table 6 on page 112.

UEP_FC_RECORD_ID_L
Address of the halfword value of KEYLENGTH, which is the (possibly partial) length of the record identifier.

The length of the record identifier is an optional input parameter on READ, WRITE, DELETE, START BR, READ NEXT, READ PREV, and RESET BR requests.

UEP_FC_FULL_RECORD_ID_L
Address of the halfword value of the full length of the record identifier. (The full length of the record identifier corresponds to the KEYLENGTH keyword of the EXEC CICS interface.)

The full length of the record identifier is returned as a mandatory output field on READ NEXT and READ PREV requests.

UEP_FC_RECORD_ID_TYPE
Address of a byte containing (for READ, WRITE, DELETE, START BR, READ NEXT, READ PREV, and RESET BR requests) the RIDFLD type.

UEP_FC_KEY
VSAM KSDS or AIX PATH access

UEP_FC_RBA
VSAM ESDS or KSDS via RBA access

UEP_FC_RRN
VSAM RRDS access

UEP_FC_DEBKEY
BDAM deblocking by key (READ, DELETE, START BR, and RESET BR requests only)

UEP_FC_DEBREC
BDAM deblocking by relative record (READ, DELETE, START BR, and RESET BR requests only)

UEP_FC_XRBA
VSAM extended ESDS access

UEP_FC_REQID
Address (for START BR, READ NEXT, READ PREV, RESET BR, and END BR requests) of the halfword value of REQID.

UEP_FC_NUMREC
Address of the fullword value of NUMREC, in which (if the request is DELETE with RIDFLD) the number of records that have been deleted is returned.

UEP_FC_KEY_COMPARISON
Address of a byte containing (for READ, START BR, and RESET BR requests) the key comparison setting.

UEP_FC_EQUAL
Key-equal-to comparison.

UEP_FC_GTEQ
Key-greater-than-or-equal-to comparison.

UEP_FC_GENERIC
Address of a byte containing (for READ, DELETE, START BR, and RESET BR requests) the generic key setting.

UEP_FC_GENERIC_KEY
Generic key used for key search.

UEP_FC_FULL_KEY
Full key used for key search.

UEP_FC_MASS_INSERT
Address of a byte containing (for WRITE requests) the mass insert setting.

UEP_FC_SEQUENTIAL_WRITE
Sequential mode.

UEP_FC_DIRECT_WRITE
Direct mode.

UEP_FC_READ_INTEGRITY
Address of a byte containing (for non-update READ, READ NEXT, and READ PREV requests) the read integrity setting. (In current versions of CICS, this setting applies only to VSAM RLS.)

UEP_FC_CR
Consistent read integrity.

UEP_FC_FCT_VALUE
Read integrity according to the setting in the FILE definition.

UEP_FC_NRI
No read integrity.

UEP_FC_RR
Repeatable read integrity.

UEP_FC_TOKEN
Address of a fullword containing the value of TOKEN.

If the request is READ, READ NEXT, or READ PREV with update, and the address is not null, the area is an output field in which the token is returned.

If the request is REWRITE, DELETE without RIDFLD, or UNLOCK, the area is an input field.

UEP_FC_SYSID
Address of a 4-byte area that contains the SYSID identifying the remote region. On input to the XFCFRIN exit, the area contained either:
- The value of the SYSID option of the API call, *or*
- Blanks (if SYSID was not specified).

If the XFCFRIN exit redirected the request to a different region, the area contains the SYSID of the remote region.

UEP_FC_LENGTH_ERROR_CODE
Address of a 1-byte area containing the length error code returned after the request completed. The possible values are:
- UEP_FC_LENGTH_OK
- UEP_FC_ BUFFER_LEN_TOO_SMALL
- UEP_FC_RECORD_LEN_TOO_LARGE
- UEP_FC_BUFFER_LEN_NOT_FILE_LEN
- UEP_FC_RECORD_LEN_NOT_FILE_LEN

UEP_FC_DUPLICATE_KEY_CODE
Address of a 1-byte area indicating whether the request found more than one record for the supplied key. The possible values are:
- UEP_FC_DUPLICATE KEY
- UEP_FC_NOT_DUPLICATE KEY

UEP_FC_ACCMETH_RETURN_CODE
Address of a 4-byte area in which access-method-dependent information is returned when either of the responses UEP_FC_REASON_ACCMETH_REQUEST_ERROR or UEP_FC_REASON_IO_ERROR is returned.

UEP_FC_RESPONSE
Address of a 1-byte area containing the response after the request completed:
- UEP_FC_RESPONSE_OK
- UEP_FC_RESPONSE_EXCEPTION
- UEP_FC_RESPONSE_DISASTER
- UEP_FC_RESPONSE_INVALID
- UEP_FC_RESPONSE_PURGED

UEP_FC_REASON
Address of a 1-byte area containing, if the request completed with an EXCEPTION response, the reason. The possible reasons are:
- UEP_FC_REASON_ABEND
- UEP_FC_REASON_ACCMETH_REQUEST_ERROR
- UEP_FC_REASON_BDAM_DELETE
- UEP_FC_REASON_BDAM_LENGTH_CHANGE
- UEP_FC_REASON_BDAM_KEY_CONVERSION
- UEP_FC_REASON_BDAM_READ_PREVIOUS
- UEP_FC_REASON_BDAM_WRITE_MASS_INSERT
- UEP_FC_REASON_BROWSE_UPD_NOT_RLS
- UEP_FC_REASON_CACHE_FAILURE
- UEP_FC_REASON_CFDT_CONNECT_ERROR
- UEP_FC_REASON_CFDT_DISCONNECT_ERROR
- UEP_FC_REASON_CFDT_INVALID_CONTINUATION
- UEP_FC_REASON_CFDT_POOL_FULL

- UEP_FC_REASON_CFDT_REOPEN_ERROR
- UEP_FC_REASON_CFDT_SERVER_NOT_AVAILABLE
- UEP_FC_REASON_CFDT_SERVER_NOT_FOUND
- UEP_FC_REASON_CFDT_SYSIDERR
- UEP_FC_REASON_CFDT_TABLE_GONE
- UEP_FC_REASON_CHANGED
- UEP_FC_REASON_CR_NOT_RLS
- UEP_FC_REASON_DATASET_BEING_COPIED
- UEP_FC_REASON_DEADLOCK_DETECTED
- UEP_FC_REASON_DELETE_AFTER_READ_UPDATE
- UEP_FC_REASON_DELETE_BEFORE_READ_UPDATE
- UEP_FC_REASON_DISASTER_PERCOLATION
- UEP_FC_REASON_DUPLICATE_READ_UPDATE
- UEP_FC_REASON_DUPLICATE_RECORD
- UEP_FC_REASON_DUPLICATE_REQID
- UEP_FC_REASON_END_OF_FILE
- UEP_FC_REASON_ESDS_DELETE
- UEP_FC_REASON_FILE_DISABLED
- UEP_FC_REASON_FILE_NOT_OPEN
- UEP_FC_REASON_FILE_NOT_RECOVERABLE
- UEP_FC_REASON_FILE_NOT_FOUND
- UEP_FC_REASON_FULL_KEY_WRONG_LENGTH
- UEP_FC_REASON_GENERIC_DELETE_NOT_KSDS
- UEP_FC_REASON_GENERIC_KEY_TOO_LONG
- UEP_FC_REASON_ILLEGAL_KEY_TYPE_CHANGE
- UEP_FC_REASON_INSUFFICIENT_SPACE
- UEP_FC_REASON_INVALID_UPDATE_TOKEN
- UEP_FC_REASON_IO_ERROR
- UEP_FC_REASON_ISCINVREQ
- UEP_FC_REASON_ISC_NOT_SUPPORTED
- UEP_FC_REASON_KEY_LENGTH_NEGATIVE
- UEP_FC_REASON_KEY_STOLEN
- UEP_FC_REASON_KSDS_AND_XRBA
- UEP_FC_REASON_LOADING
- UEP_FC_REASON_LOCKED
- UEP_FC_REASON_LOST_LOCKS
- UEP_FC_REASON_LOCK_STRUCTURE_FULL
- UEP_FC_REASON_NOT_IN_SUBSET
- UEP_FC_REASON_NO_VARIABLE_LENGTH
- UEP_FC_REASON_NOSUSPEND_NOT_RLS
- UEP_FC_REASON_NOTAUTH
- UEP_FC_REASON_NOT_EXTENDED
- UEP_FC_REASON_PREVIOUS_RLS_FAILURE
- UEP_FC_REASON_RBA_ACCESS_TO_RLS_KSDS
- UEP_FC_REASON_READ_NOT_AUTHORISED
- UEP_FC_REASON_READPREV_IN_GENERIC_BROWSE
- UEP_FC_REASON_RECLEN_EXCEEDS_LOGGER_BFSZ
- UEP_FC_REASON_RECORD_BUSY
- UEP_FC_REASON_RECORD_NOT_FOUND
- UEP_FC_REASON_REMOTE_INVREQ
- UEP_FC_REASON_RESTART_FAILED
- UEP_FC_REASON_REWRITE_BEFORE_READ_UPDATE
- UEP_FC_REASON_RIDFLD_KEY_NOT_RECORD_KEY
- UEP_FC_REASON_RLS_DEADLOCK_DETECTED
- UEP_FC_REASON_RLS_DISABLED
- UEP_FC_REASON_RLS_FAILURE

- UEP_FC_REASON_RR_NOT_RLS
- UEP_FC_REASON_SECURITY_FAILURE
- UEP_FC_REASON_SELF_DEADLOCK_DETECTED
- UEP_FC_REASON_SERVREQ_VIOLATION
- UEP_FC_REASON_SHIP
- UEP_FC_REASON_SHIPPED_SECURITY_FAILURE
- UEP_FC_REASON_STORE_FAIL
- UEP_FC_REASON_SUPPRESSED
- UEP_FC_REASON_SYSIDERR
- UEP_FC_REASON_TABLE_FULL
- UEP_FC_REASON_TABLE_TOKEN_INVALID
- UEP_FC_REASON_TIMEOUT
- UEP_FC_REASON_TOO_MANY_CFDTS_IN_UOW
- UEP_FC_REASON_UNKNOWN_REQID_ENDBR
- UEP_FC_REASON_UNKNOWN_REQID_READNEXT
- UEP_FC_REASON_UNKNOWN_REQID_READPREV
- UEP_FC_REASON_UNKNOWN_REQID_RESETBR
- UEP_FC_REASON_UPDATE_NOT_AUTHORISED

UEP_FC_EXIT_TOKEN
Address of the 4-byte token passed from XFCFRIN.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

API and SPI calls
None can be used.

File control EXEC interface API exits XFCREQ and XFCREQC

The XFCREQ exit allows you to intercept a file control application programming interface (API) request before any action has been taken on it by file control. The XFCREQC exit allows you to intercept a file control API request after file control has completed its processing.

Important

The XFCREQ and XFCREQC exits are not called, on the target region, for function-shipped requests. That is, if a file control API request is function-shipped to a remote region, the exits are not called on the remote region. To intercept a function-shipped file control API request on the target region, use the XFCFRIN exit—see "File control domain exits, XFCFRIN and XFCFROUT" on page 93.

The file control API commands intercepted are:

- READ
- WRITE
- REWRITE
- DELETE
- UNLOCK
- STARTBR

- READNEXT
- READPREV
- ENDBR
- RESETBR.

The XFCREQ and XFCREQC exits can be written only in assembler language.

Using XFCREQ, you can:

- Analyze the request, to determine its type, the keywords specified, and their values.
- Modify values specified by the request before the command is executed.
- Set return codes to specify that either:
  - CICS should continue with the (possibly modified) request.
  - CICS should bypass the request. (Note that if you set this return code, you must also set up return codes for the EXEC interface block (EIB), as if you had processed the request yourself.)

Using XFCREQC, you can:

- Analyze the request, to determine its type, the keywords specified, and their values.
- Set return codes for the EIB.

Both exits are passed nine parameters as follows:

- The address of the command-level parameter structure
- The address of a token (UEPFCTOK) used to pass 4 bytes of data from XFCREQ to XFCREQC
- The addresses of copies of four pieces of return code and resource information from the EIB
- The address of a token (UEPTSTOK) that is valid throughout the life of a task
- The address of a recursion count field
- The address of a 16-byte area that is used if the request has been function shipped.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The command-level parameter structure

The command-level parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of a bit string that describes the type of request and identifies each keyword specified with the

request. The remaining addresses point to pieces of data associated with the request. For example, the second address always points to the file name.

Only the first 8 addresses and the last address can be referenced by the user exit. The ninth through eleventh addresses are reserved for CICS internal use.

You can examine the EID to determine the type of request and the keywords specified. You can examine the other parameters in the list to determine the values of the keywords. You can also modify values of keywords specified on the request. (For example, you could change the name of the file involved in the request.)

End of parameter list indicator

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

The original parameter list, as it was before XFCREQ was invoked, is restored after the completion of XFCREQC. It follows that the execution diagnostic facility (EDF) displays the original command before and after execution.

Note: EDF does not display any changes made by the exit.

The UEPCLPS exit-specific parameter:

The UEPCLPS exit-specific parameter is the address of the command-level parameter structure, and is included in exits XFCREQ and XFCREQC.

The command-level parameter structure contains 12 addresses, FC_ADDR0 through FC_ADDRB. It is defined in the DSECT FC_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHFCEDS.

The command-level parameter list is made up as follows:

FC_ADDR0
is the address of a 9-byte area called the EID, which is made up as follows:

- **FC_GROUP**
- **FC_FUNCT**
- **FC_BITS1**
- **FC_BITS2**
- **FC_EIDOPT5**
- **FC_EIDOPT6**
- **FC_EIDOPT7**
- **FC_EIDOPT8**

The name of the DSECT mapping the EID is FC_EID.

FC_GROUP
Always X'06', indicating that this is a file control request.

FC_FUNCT
One byte that defines the type of request:

X'02' READ

X'04' WRITE

X'06' REWRITE

X'08' DELETE

X'0A' UNLOCK

X'0C' STARTBR

X'0E' READNEXT

X'10' READPREV

X'12' ENDBR

X'14' RESETBR

FC_BITS1

Existence bits that define which keywords that contain values were specified. To obtain the value associated with a keyword, you need to use the appropriate address from the command-level parameter structure. Before using this address, you must check the associated existence bit. If the existence bit is set off, the keyword was not specified in the request and the address should not be used.

X'80' Set if the request contains the keyword FILE. If set, **FC_ADDR1** is meaningful.

X'40' Set if the request contains any of the keywords INTO, SET, or FROM. If set, **FC_ADDR2** is meaningful.

X'20' Set if the request specifies LENGTH or NUMREC, or if a STARTBR, RESETBR, or ENDBR request specifies REQID. If set, **FC_ADDR3** is meaningful.

X'10' Set if the request specifies RIDFLD. If set, **FC_ADDR4** is meaningful.

X'08' Set if the request specifies KEYLENGTH. If set, **FC_ADDR5** is meaningful.

X'04' Set if the request is READNEXT or READPREV and specifies REQID. If set, **FC_ADDR6** is meaningful.

X'02' Set if the request specifies SYSID. If set, **FC_ADDR7** is meaningful.

X'01' Not used by file control.

FC_BITS2

Second set of existence bits.

X'20' Set if the request specifies TOKEN. If set, FC_ADDRB is meaningful.

FC_EIDOPT5

Indicates whether certain keywords that do not take values were specified on the request.

X'04' MASSINSERT specified.

X'02' RRN specified.

X'01' SET (and not INTO) was specified.

Note: Your program must test for keywords at the bit level, because there may be more than one of these keywords present.

FC_EIDOPT6
Indicates whether certain keywords that do not take values were specified on the request.

X'80' RBA specified.

X'40' GENERIC specified.

X'20' GTEQ specified.

X'10' UNCOMMITTED specified.

X'08' CONSISTENT specified.

X'04' REPEATABLE specified.

X'01' NOSUSPEND specified (on READ, READNEXT, READPREV, WRITE, DELETE, or REWRITE).

Note:

1. If the read integrity bits (for UNCOMMITTED, CONSISTENT, and REPEATABLE) are off (zero) on the command, the read integrity options specified on the file resource definition are used. If you need to know what these are, you can issue an EXEC CICS INQUIRE FILE command.
2. Your program must test for keywords at the bit level, because there may be more than one of these keywords present.

FC_EIDOPT7
Indicates whether certain keywords that do not take values were specified on the request.

X'04' UPDATE specified. This setting is meaningful only for READ requests. For other requests, X'04' may or may not be set.

X'01' Either DEBREC or DEBKEY specified (see **FC_EIDOPT8**). This setting is meaningful only for READ requests. For other requests, X'01' may or may not be set.

Note: Your program must test for keywords at the bit level, because there may be more than one of these keywords present.

FC_EIDOPT8
Indicates whether certain keywords that do not take values were specified on the request.

X'80' DEBKEY specified.

X'40' DEBREC specified.

X'20' TOKEN specified.

X'08' XRBA specified. If the XRBA bit is on, FC_RIDFLD (described in DSECT DFHFCEDS) points to an 8-byte extended relative byte address (XRBA).

FC_ADDR1
is the address of an 8-byte area containing the name specified on the FILE keyword.

FC_ADDR2
is the address of one of the following:

- A 4-byte address returned for SET (if the request is READ, READNEXT, or READPREV, and if **FC_EIDOPT5** indicates that this is SET).
- Data returned for INTO (if the request is READ, READNEXT, or READPREV, and if **FC_EIDOPT5** indicates that this is not SET).
- Data from FROM (if the request is WRITE or REWRITE).

FC_ADDR3
is the address of one of the following:

- The halfword value of LENGTH (if the request is READ, WRITE, REWRITE, READNEXT, or READPREV).

  Warning: For requests that specify INTO, do not change the value of LENGTH to a value greater than that specified by the application. To do so causes a storage overlay in the application.
- The returned halfword value of NUMREC (if the request is DELETE).
- The halfword value of REQID (if the request is STARTBR, RESETBR, or ENDBR).

FC_ADDR4
is the address of an area containing the value of the RIDFLD keyword.

FC_ADDR5
is the address of the halfword value of KEYLENGTH.

FC_ADDR6
is the address of the halfword value of REQID (if the request is READNEXT or READPREV).

FC_ADDR7
is the address of an area containing the value of SYSID.

FC_ADDR8
is the address of a value intended for CICS internal use only. It must not be used.

FC_ADDR9
is the address of a value intended for CICS internal use only. It must not be used.

FC_ADDRA
is the address of a value intended for CICS internal use only. It must not be used.

FC_ADDRB
is the address of the fullword value of TOKEN (if the request is READ, READNEXT, READPREV, REWRITE, DELETE, or UNLOCK).

Modifying fields in the command-level parameter structure

Some fields that are passed to file control are used as input to the request, some are used as output fields, and some are used for both input and output. The method your user exit program uses to modify a field depends on the usage of the field.

A list of input and output fields:
The following are always input fields:

- FILE
- FROM

- KEYLENGTH
- REQID
- SYSID

The following are always output fields:
- INTO
- NUMREC
- SET

Whether LENGTH and RIDFLD are input or output fields depends on the request, as shown in Table 6. A dash (—) means that the keyword cannot be specified on the request.

Table 6. LENGTH and RIDFLD as input and output fields

Request	LENGTH	RIDFLD
READ	Output	See Note 1.
WRITE	Input	See Note 2.
REWRITE	Input	—
DELETE	—	See Note 3.
UNLOCK	—	—
STARTBR	—	Input
READNEXT	Output	Output
READPREV	Output	Output
ENDBR	—	—
RESETBR	—	Input

Note:
1. Normally, this is an input field. However, if UPDATE is specified and the file is a BDAM file using extended key search, RIDFLD is used for both input and output.
2. The use of RIDFLD on a WRITE request depends on the file type. For a VSAM KSDS or RRDS, or a fixed-format BDAM file, RIDFLD is an input field. For all other file types, it is used either for output only, or for both input and output, and should be treated like an output field.
3. RIDFLD is an input field on DELETE requests that are not preceded by a READ UPDATE. It is not specified on requests that are preceded by a READ UPDATE.

Modifying input fields:

The correct method of modifying an input field is to create a new copy of it, and to change the address in the command-level parameter list to point to your new data.

Note: You must never modify an input field by altering the data that is pointed to by the command-level parameter list. To do so would corrupt storage belonging to the application program and would cause a failure when the program attempted to reuse the field.

Modifying output fields: The technique described in "Modifying input fields" on page 112 is not suitable for modifying output fields. (The results would be returned to the new area instead of the application's area, and would be invisible to the application.)

An output field is modified by altering the data that is pointed to by the command-level parameter list. In the case of an output field, you can modify the application's data in place, because the application is expecting the field to be modified anyway.

Modifying fields used for both input and output:

An example of a field that is used for both input and output is LENGTH on a READ request that specifies INTO. You can treat such fields in the same way as output fields, and they are considered to be the same.

Modifying the EID

It is not possible to modify the EID to make major changes to requests. It is not possible, for example, to change a WRITE request to a READ request. However, you can make minor changes to requests, such as to turn on the existence bit for SYSID so that the request can be changed into one that is shipped to a remote system.

The list that follows shows the bits in the EID that can be modified. Any attempt to modify any other part of the EID is ignored.

FC_BITS1

- **X'20'** The existence bit for LENGTH, NUMREC, or (if the request is STARTBR, RESETBR, or ENDBR) REQID.
- **X'08'** The existence bit for KEYLENGTH.
- **X'04'** The existence bit for REQID if the request is READNEXT or READPREV.
- **X'02'** The existence bit for SYSID.

FC_BITS2

- **X'20'** Token specified.

FC_EIDOPT5

- **X'04'** MASSINSERT specified.

FC_EIDOPT6

- **X'40'** GENERIC specified.
- **X'20'** GTEQ specified.
- **X'10'** UNCOMMITTED specified.
- **X'08'** CONSISTENT specified.
- **X'04'** REPEATABLE specified.
- **X'02'** UPDATE specified on READNEXT or READPREV.
- **X'01'** NOSUSPEND specified (on READ, READNEXT, READPREV, WRITE, DELETE, or REWRITE).

Bits in the EID should be modified in place. You should not modify the pointer to the EID: any attempt to do so is ignored by CICS.

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the file control request only.

If more than one of UNCOMMITTED, CONSISTENT, or REPEATABLE is specified, CONSISTENT takes precedence over UNCOMMITTED, and REPEATABLE takes precedence over CONSISTENT and UNCOMMITTED.

Example of modifying read integrity bits:

You might want all RLS read requests from all programs against a specific file to specify CONSISTENT read. You could code a user exit program that turns on the bit for CONSISTENT and turns off the other two read integrity bits in all requests to the file. You could partially achieve this effect by specifying CONSISTENT on the FILE definition. However, that would only override requests that did not explicitly specify a level of read integrity. Using a global user exit program for this purpose also overrides programs that explicitly specify UNCOMMITTED or REPEATABLE.

Warnings:

1. If a global user exit program changes a file request to request a higher level of read integrity (for example, it changes the request from UNCOMMITTED to REPEATABLE), this could cause CICS either to acquire extra read locks, or to keep its read locks for a longer period of time. This may degrade system throughput, by causing other transactions to wait, or introduce deadlocks.
2. If a global user exit program changes the request to one that requests a lower level of read integrity (for example, it changes the request from REPEATABLE to UNCOMMITTED), this could cause application logic errors to occur in the program that originated the request. The errors could occur because the application program may be relying on the record to remain unchanged while it reads a series of other, related, records. This can be guaranteed with REPEATABLE, but not if the option is changed to UNCOMMITTED.
3. Your user exit program is prevented from making major changes to the EID. However, you must take great care when making the minor modifications that **are** permitted. For instance, it is possible to change a DELETE into a GENERIC DELETE, but to make such a change may be dangerous.

Use of the parameter UEPFSHIP

UEPFSHIP contains the address of a 16-byte area. This area consists of 4 characters, followed by 3 fullwords.

If the first byte contains 'Y', this request has been function shipped to this region. In this case, if your exit program wants to bypass file control (by setting a return code of UERCBYP), it must set the 3 fullwords as follows:

Fullword 1
The length of the buffer area

Fullword 2
The length of the record

Fullword 3
The length of the modified RIDFLD.

Doing this ensures that the data and RIDFLD are correctly shipped back.

EIB (EXEC interface block)

Copies of EIBRSRCE, EIBRCODE, EIBRESP, and EIBRESP2 are passed to the exit so you can modify/set the completion and resource information in XFCREQ and XFCREQC, and examine completion and resource information in XFCREQC.

You can update the copies of EIBRSRCE, EIBRCODE, EIBRESP, and EIBRESP2 that you are given in the parameter list. File Control copies your values into the real EIB after the completion of XFCREQC; or if you specify a return code of 'bypass' in XFCREQ.

You must set valid file control responses. You must set all three of EIBRCODE, EIBRESP, and EIBRESP2 to a consistent set of values, such as would be set by File Control to describe a valid completion. **File Control does not police the consistency of EIBRCODE, EIBRESP, and EIBRESP2**. To aid you in setting the values of EIBRCODE, EIBRESP, and EIBRESP2, the values used by File Control are specified in DFHFCEDS.

Example of how XFCREQ and XFCREQC can be used

In this example, XFCREQ and XFCREQC are used to obtain a record containing compressed data, to decompress the data, and to return it to the area specified by the user program as INTO. The example shows only the capabilities of the exits; it is not intended to indicate an ideal way of achieving the function.

In XFCREQ:

1. Issue an EXEC CICS GETMAIN to obtain an area large enough to hold the decompressed data.
2. Change the INTO pointer to point to this new area, so that File Control uses it when it processes the request. (The decompressed data is copied to the *user's* INTO area, and the INTO pointer reset, before return to the application program—see stages 4 and 7 of the processing to be done by XFCREQC.)
3. Set UEPFCTOK to be the address of the new area so that XFCREQC can also use this area.
4. Return to CICS.

In XFCREQC:

1. Check 'UEPRCODE' to make sure that the file control request completed without error.
2. Use UEPFCTOK to find the address of the area. This area now holds the compressed data.
3. Decompress the data in place.
4. Copy the data from the new area to the user's INTO area. Use the user-specified LENGTH (from the command-level parameter list) to ensure that the data fits and that the copy does not cause a storage violation.
5. Set 'LENGERR' in UEPRESP, UEPRESP2, and UEPRCODE if the data does not fit.
6. Use EXEC CICS FREEMAIN to free the work area pointed to by UEPFCTOK.
7. At this point the command-level parameter list points to the now free area as the address for INTO. This is not a problem, because after completion of XFCREQC File Control restores this pointer to point to the area supplied by the user program.
8. Return to CICS.

Exit XFCREQ

Exit XFCREQ is invoked before CICS processes a file control API request. The exit is not invoked, on the target region, for function-shipped requests.

Exit-specific parameters

UEPCLPS
Address of the command-level parameter structure. See "The UEPCLPS exit-specific parameter" on page 108.

UEPFCTOK
Address of the 4-byte token to be passed to XFCREQC. This allows you, for example, to pass a work area to exit XFCREQC.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code 'EIBRCODE'. For details of EIB return codes, refer to EIB fields, in the *CICS Application Programming Reference* manual.

UEPRESP
Address of a 4-byte binary copy of the EIB response code 'EIBRESP'.

UEPRESP2
Address of a 4-byte binary copy of the EIB response code 'EIBRESP2'.

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPFSHIP
Address of a 16 byte area. See "Use of the parameter UEPFSHIP" on page 114.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

Return codes

UERCNORM
Continue processing.

UERCBYP
The file control EXEC interface program should ignore this request.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.

API and SPI calls
All can be used, except for:

EXEC CICS SHUTDOWN
EXEC CICS XCTL

Note:

1. Take care when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a file control request is issued from the XFCREQ exit. Use of the recursion counter UEPRECUR is recommended.
2. Exit programs that issue EXEC CICS commands must first address the EIB. See "Using CICS services" on page 4.
3. Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See "Returning values to CICS" on page 10.

Exit XFCREQC

Exit XFCREQC is invoked after a file control API request has completed, and before return from the file control EXEC interface program. The exit is not invoked, on the target region, for function-shipped requests.

Exit-specific parameters

UEPCLPS
Address of the command-level parameter structure. See "The UEPCLPS exit-specific parameter" on page 108.

UEPFCTOK
Address of the 4 byte token passed from XFCREQ.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code 'EIBRCODE'. For details of EIB return codes, refer to EIB fields, in the *CICS Application Programming Reference* manual.

UEPRESP
Address of a 4-byte binary copy of the EIB response code 'EIBRESP'.

Note: If the file that has just been accessed is remote, the addressed field contains zeros (even if UEPRCODE is non-zero).

UEPRESP2
Address of a 4-byte binary copy of the EIB response code 'EIBRESP2'.

Note: If the file that has just been accessed is remote, the addressed field contains zeros (even if UEPRCODE is non-zero).

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.

API and SPI calls
All can be used, except for:
EXEC CICS SHUTDOWN
EXEC CICS XCTL

Note:

1. Take care when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a file control request is issued from the XFCREQC exit. Use of the recursion counter UEPRECUR is recommended.
2. Exit programs that issue EXEC CICS commands must first address the EIB. See "Using CICS services" on page 4.
3. Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See "Returning values to CICS" on page 10.

Example program

CICS supplies, in CICSTS52.CICS.SDFHSAMP, an example program, DFH$XTSE, that shows how to modify fields in the command-level parameter structure passed to EXEC interface exits. DFH$XTSE is listed in Appendix G, "The example program for the XTSEREQ global user exit, DFH$XTSE," on page 917.

File control EXEC interface SPI exits XFCAREQ and XFCAREQC

The XFCAREQ exit allows you to intercept a file control system programming interface (SPI) request before any action has been taken on it by file control. The XFCAREQC exit allows you to intercept the response after a file control SPI request has completed.

The file control SPI requests intercepted are:

- **EXEC CICS INQUIRE FILE**
- **EXEC CICS SET FILE**.

Using XFCAREQ, you can:

- Analyze the SPI parameter list (function, keywords, argument values, and responses)
- Modify any input parameter before execution of the request
- Prevent execution of a request and set appropriate responses.

Using XFCAREQC, you can:

- Analyze the SPI parameter list
- Modify any output parameter value and set responses after execution.

You can also:

- Pass data between your XFCAREQ and XFCAREQC exit programs when they are invoked for the same request.
- Pass data between your file control exit programs when they are invoked within the same task. You can pass data between successive invocations of XFCAREQ and XFCAREQC and also between invocations of other EXEC-enabled user exits.

If you make changes to file states (that is, if you open, close, enable, or disable a file) it is possible that exits in the file state change program (XFCSREQ and XFCSREQC) could modify situations set up by XFCAREQ. Therefore you must consider the order in which the exits are invoked. If all four exits are enabled, the order of invocation is as follows:

- For the **SET FILE** command:
  1. XFCAREQ
  2. XFCSREQ
  3. XFCSREQC
  4. XFCAREQC
- For the **INQUIRE FILE** command:
  1. XFCAREQ
  2. XFCAREQC

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XFCAREQ

This exit is invoked before CICS processes a file control SPI request.

When invoked

Before CICS processes a file control SPI request.

Exit-specific parameters

UEPCLPS

Address of a copy of the SPI command parameter list. See "The command-level parameter structure" on page 121.

UEPFATOK

Address of a 4-byte area that can be used to pass information between XFCAREQ and XFCAREQC on a single file control SPI request.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code EIBRCODE. For details of EIB return codes, see EIB fields, in the *CICS Application Programming Reference* manual.

UEPRESP
Address of a 4-byte binary copy of the EIB response code EIBRESP.

UEPRESP2
Address of a 4-byte binary copy of the EIB response code EIBRESP2.

UEPTSTOK
Address of a 4-byte token which can be used to pass information between successive file control requests within the same task (for example, between successive invocations of the XFCAREQC exit). See "Using the task token UEPTSTOK" on page 19.

UEPRECUR
Address of a halfword recursion counter. The counter is set to zero when the exit is first invoked and is incremented for each recursive call.

Return codes

UERCBYP
Bypass this request.

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI commands
All can be used.

API and SPI commands
All can be used, except for:
EXEC CICS SHUTDOWN
EXEC CICS XCTL

Note: Take care when using recursive commands. For example, you must avoid entering a loop when issuing a file control SPI request from the XFCAREQ exit. Use of the recursion counter UEPRECUR is recommended.

Exit XFCAREQC

Exit XFCAREQC is invoked after a file control SPI request has completed, before return from the file control SPI EXEC interface program.

Exit specific parameters:

UEPCLPS
Address of a copy of the API command parameter list. See "The command-level parameter structure" on page 121.

UEPFATOK
Address of a 4-byte area that can be used to pass information between XFCAREQ and XFCAREQC on a single file control SPI request.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code

EIBRCODE. For details of EIB return codes, see EIB fields, in the *CICS Application Programming Reference* manual.

UEPRESP
Address of a 4-byte binary copy of the EIB response code EIBRESP.

UEPRESP2
Address of a 4-byte binary copy of the EIB response code EIBRESP2.

UEPTSTOK
Address of a 4-byte token which can be used to pass information between successive file control requests within the same task (for example, between successive invocations of the XFCAREQC exit). See "Using the task token UEPTSTOK" on page 19.

UEPRECUR
Address of a halfword recursion counter. The counter is set to zero when the exit is first invoked and is incremented for each recursive call.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI commands
All can be used.

API and SPI commands
All can be used, except for:
EXEC CICS SHUTDOWN
EXEC CICS XCTL

You can update the copies of EIBRCODE, EIBRESP, and EIBRESP2 that you are given in the parameter list. If you update the values, file control copies the new values into the application program's EXEC interface block (EIB) after the completion of XFCAREQC or if you specify a return code of UERCBYP in XFCAREQ.

You must set valid file control responses. You must set all three of EIBRCODE, EIBRESP, and EIBRESP2 to a consistent set of values, such as would be set by file control to describe a valid completion. CICS does not check the consistency of the values you set. If EIBRCODE is set to a non-zero value and EIBRESP is set to zero, CICS overrides EIBRESP with a non-zero value. To help you set values for EIBRCODE, EIBRESP, and EIBRESP2, the values used by file control for SPI requests are specified in DSECT DFHFAUED.

Note: Take care when using recursive commands. For example, you must avoid entering a loop when issuing a file control SPI request from the XFCAREQ exit. Use of the recursion counter UEPRECUR is recommended.

The command-level parameter structure

The command-level parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of a bit string that describes the type of request and identifies each keyword specified with the request. The remaining addresses point to pieces of data associated with the request.

You can examine the EID to determine the type of request and the keywords specified. You can examine the other parameters in the list to determine the values of the keywords. You can also modify values of keywords specified on the request.

Note: The relationship between arguments, keywords, data types, and input/output types on the file control SPI commands is summarized in the following tables:

- For INQUIRE FILE, see Table 7 on page 128.
- For SET FILE, see Table 8 on page 130.

The UEPCLPS exit-specific parameter:

The UEPCLPS exit-specific parameter is passed to both XFCAREQ and XFCAREQC and contains the address of the command-level parameter structure.

The command-level parameter list contains 64 addresses, FCIS_ADDR0 through FCIS_ADDR63. These are described in DSECT DFHFAUED, which you should copy into your program by including the statement COPY DFHFAUED.

The command-level parameter list is made up as follows:

FCIS_ADDR0
is the address of an 13-byte area called the EID which is made up as follows:

- **FCIS_GROUP**
- **FCIS_FUNCT**
- **FCIS_EIDOPT2**
- **FCIS_EIDOPT3**
- **FCIS_EIDOPT4**
- **FCIS_BITS1**
- **FCIS_BITS2**
- **FCIS_BITS3**
- **FCIS_BITS4**
- **FCIS_BITS5**
- **FCIS_BITS6**
- **FCIS_BITS7**
- **FCIS_BITS8**

FCIS_GROUP
Always X'4C', indicating that this is a file control SPI request.

FCIS_FUNCT
One byte that defines the type of request:

X'02' INQUIRE FILE

X'04' SET FILE.

FCIS_EIDOPT2
Not used by file control.

FCIS_EIDOPT3
Not used by file control.

FCIS_EIDOPT4
Not used by file control.

FCIS_BITS1

Existence bits which specify which arguments were specified. To obtain the argument associated with a keyword, you need to obtain the appropriate address from the command-level parameter structure. Before using this address you must check the associated existence bit. If the existence bit is set off, the argument was not specified in the request and the address should not be used.

X'80' Set if the request contains an argument for the FILE keyword. If set, FCIS_ADDR1 is meaningful.

X'40' Set if the request contains an argument for the DSNAME keyword. If set, FCIS_ADDR2 is meaningful.

X'20' Set if the request contains an argument for the FWDRECSTATUS keyword. If set, FCIS_ADDR3 is meaningful.

X'10' Set if the request contains an argument for the STRINGS keyword. If set, FCIS_ADDR4 is meaningful.

X'08' Set if the request contains an argument for the BASEDSNAME keyword. If set, FCIS_ADDR5 is meaningful.

X'04' Set if the request contains an argument for the LSRPOOLID keyword. If set, FCIS_ADDR6 is meaningful.

X'02' Set if the request contains an argument for the READ keyword. If set, FCIS_ADDR7 is meaningful.

X'01' Set if the request contains an argument for the UPDATE keyword. If set, FCIS_ADDR8 is meaningful.

FCIS_BITS2

Existence bits which specify which arguments were specified. The comments below FCIS_BITS1 also apply to FCIS_BITS2.

X'80' Set if the request contains an argument for the BROWSE keyword. If set, FCIS_ADDR9 is meaningful.

X'40' Set if the request contains an argument for the ADD keyword. If set, FCIS_ADDR10 is meaningful.

X'20' Set if the request contains an argument for the DELETE keyword. If set, FCIS_ADDR11 is meaningful.

X'10' Set if the request contains an argument for the DISPOSITION keyword. If set, FCIS_ADDR12 is meaningful.

X'08' Set if the request contains an argument for the EMPTYSTATUS keyword. If set, FCIS_ADDR13 is meaningful.

X'04' Set if the request contains an argument for the OPENSTATUS keyword. If set, FCIS_ADDR14 is meaningful.

X'02' Set if the request contains an argument for the ENABLESTATUS keyword. If set, FCIS_ADDR15 is meaningful.

X'01' Set if the request contains an argument for the RECOVSTATUS keyword. If set, FCIS_ADDR16 is meaningful.

FCIS_BITS3

Existence bits which specify which arguments were specified. The comments below FCIS_BITS1 also apply to FCIS_BITS3.

X'80' Set if the request contains an argument for the ACCESSMETHOD keyword. If set, FCIS_ADDR17 is meaningful.

X'40' Set if the request contains an argument for the TYPE keyword. If set, FCIS_ADDR18 is meaningful.

X'20' Set if the request contains an argument for the OBJECT keyword. If set, FCIS_ADDR19 is meaningful.

X'10' Set if the request contains an argument for the REMOTESYSTEM keyword. If set, FCIS_ADDR20 is meaningful.

X'08' Set if the request contains an argument for the REMOTENAME keyword. If set, FCIS_ADDR21 is meaningful.

X'04' Set if the request contains an argument for the RECORDFORMAT keyword. If set, FCIS_ADDR22 is meaningful.

X'02' Set if the request contains an argument for the BLOCKFORMAT keyword. If set, FCIS_ADDR23 is meaningful.

X'01' Set if the request contains an argument for the KEYLENGTH keyword. If set, FCIS_ADDR24 is meaningful.

FCIS_BITS4

Existence bits which specify which arguments were specified. The comments below FCIS_BITS1 also apply to FCIS_BITS4.

X'80' Set if the request contains an argument for the KEYPOSITION keyword. If set, FCIS_ADDR25 is meaningful.

X'40' Set if the request contains an argument for the RECORDSIZE keyword. If set, FCIS_ADDR26 is meaningful.

X'20' Set if the request contains an argument for the RELTYPE keyword. If set, FCIS_ADDR27 is meaningful.

X'10' Set if the request contains an argument for the EXCLUSIVE keyword. If set, FCIS_ADDR28 is meaningful.

X'08' Set if the request contains an argument for the BLOCKKEYLEN keyword. If set, FCIS_ADDR29 is meaningful.

X'04' Set if the request contains an argument for the BLOCKSIZE keyword. If set, FCIS_ADDR30 is meaningful.

X'02' Not used by file control.

X'01' Not used by file control.

FCIS_BITS5

Existence bits which specify which arguments were specified. The comments below FCIS_BITS1 also apply to FCIS_BITS5.

X'80' Set if the request contains an argument for the TABLE keyword. If set, FCIS_ADDR33 is meaningful.

X'40' Set if the request contains an argument for the MAXNUMRECS keyword. If set, FCIS_ADDR34 is meaningful.

X'20' Set if the request contains an argument for the READINTEG keyword. If set, FCIS_ADDR35 is meaningful.

X'10' Set if the request contains an argument for the RLSACCESS keyword. If set, FCIS_ADDR36 is meaningful.

X'08' Set if the request contains an argument for the DEFINESOURCE keyword. If set, FCIS_ADDR37 is meaningful.

X'04' Set if the request contains an argument for the INSTALLAGT keyword. If set, FCIS_ADDR38 is meaningful.

X'02' Set if the request contains an argument for the INSTALLUSR keyword. If set, FCIS_ADDR39 is meaningful.

X'01' Set if the request contains an argument for the CHANGEAGENT keyword. If set, FCIS_ADDR40 is meaningful.

FCIS_BITS6

Specifies whether certain keywords were specified on the File control SPI command.

X'80' Set if the request contains the START keyword.

X'40' Set if the request contains the NEXT keyword.

X'20' Set if the request contains the END keyword.

X'10' Set if the request contains the WAIT keyword.

X'08' Set if the request contains the NOWAIT keyword.

X'04' Set if the request contains the FORCE keyword.

X'02' Set if the request contains the ENABLED keyword.

X'01' Set if the request contains the DISABLED keyword.

FCIS_BITS7

Specifies whether certain keywords were specified on the File control SPI command. Also contains the existence bit for JOURNALNUM.

X'80' Set if the request contains the OPEN keyword.

X'40' Set if the request contains the CLOSED keyword.

X'20' Set if the request contains the EMPTY keyword.

X'10' Set if the request contains an argument for the JOURNALNUM keyword. If set, FCIS_ADDR52 is meaningful.

X'08' Set if the request contains the LOADTYPE keyword.

X'04' Set if the request contains the POOL keyword.

X'02' Set if the request contains the TABLENAME keyword.

X'01' Set if the request contains the UPDATEMODEL keyword.

FCIS_BITS8

X'80' Set if the request contains the REMOTETABLE keyword.

X'40' Not used by file control.

X'20' Set if the request contains an argument for the CHANGEUSRID keyword. If set, FCIS_ADDR59 is meaningful.

X'10' Set if the request contains an argument for the CHANGEAGREL keyword. If set, FCIS_ADDR60 is meaningful.

X'08' Set if the request contains an argument for the DEFINETIME keyword. If set, FCIS_ADDR61 is meaningful.

X'04' Set if the request contains an argument for the CHANGETIME keyword. If set, FCIS_ADDR62 is meaningful.

X'02' Set if the request contains an argument for the INSTALLTIME keyword. If set, FCIS_ADDR63 is meaningful.

X'01' Not used by file control.

FCIS_ADDR1
is the address of an 8-byte area containing the name from FILE.

FCIS_ADDR2
is the address of a 44-byte area containing the name from DSNAME.

FCIS_ADDR3
is the address of a 4-byte area containing the CVDA from FWDRECOVSTATUS.

FCIS_ADDR4
is the address of a 4-byte area containing the data from STRINGS.

FCIS_ADDR5
is the address of a 44-byte area containing the name from BASEDSNAME.

FCIS_ADDR6
is the address of a 4-byte area containing the data from LSRPOOLID.

FCIS_ADDR7
is the address of a 4-byte area containing the CVDA from READ.

FCIS_ADDR8
is the address of a 4-byte area containing the CVDA from UPDATE.

FCIS_ADDR9
is the address of a 4-byte area containing the CVDA from BROWSE.

FCIS_ADDR10
is the address of a 4-byte area containing the CVDA from ADD.

FCIS_ADDR11
is the address of a 4-byte area containing the CVDA from DELETE.

FCIS_ADDR12
is the address of a 4-byte area containing the CVDA from DISPOSITION.

FCIS_ADDR13
is the address of a 4-byte area containing the CVDA from EMPTYSTATUS.

FCIS_ADDR14
is the address of a 4-byte area containing the CVDA from OPENSTATUS.

FCIS_ADDR15
is the address of a 4-byte area containing the CVDA from ENABLESTATUS.

FCIS_ADDR16
is the address of a 4-byte area containing the CVDA from RECOVSTATUS.

FCIS_ADDR17
is the address of a 4-byte area containing the CVDA from ACCESSMETHOD.

FCIS_ADDR18
is the address of a 4-byte area containing the CVDA from TYPE.

FCIS_ADDR19
is the address of a 4-byte area containing the CVDA from OBJECT.

FCIS_ADDR20
is the address of a 4-byte area containing the name from REMOTESYSTEM.

FCIS_ADDR21
is the address of an 8-byte area containing the name from REMOTENAME.

FCIS_ADDR22
is the address of a 4-byte area containing the CVDA from RECORDFORMAT.

FCIS_ADDR23
is the address of a 4-byte area containing the CVDA from BLOCKFORMAT.

FCIS_ADDR24
is the address of a 4-byte area containing the CVDA from KEYLENGTH.

FCIS_ADDR25
is the address of a 4-byte area containing the data from KEYPOSITION.

FCIS_ADDR26
is the address of a 4-byte area containing the data from RECORDSIZE.

FCIS_ADDR27
is the address of a 4-byte area containing the CVDA from RELTYPE.

FCIS_ADDR28
is the address of a 4-byte area containing the CVDA from EXCLUSIVE.

FCIS_ADDR29
is the address of a 4-byte area containing the data from BLOCKKEYLEN.

FCIS_ADDR30
is the address of a 4-byte area containing the data from BLOCKSIZE.

FCIS_ADDR31
is not used by file control.

FCIS_ADDR32
is the address of a 4-byte area containing the data from BUSY.

FCIS_ADDR33
is the address of a 4-byte area containing the CVDA from TABLE.

FCIS_ADDR34
is the address of a 4-byte area containing the data from MAXNUMRECS.

FCIS_ADDR35
is the address of a 4-byte area containing the CVDA from READINTEG.

FCIS_ADDR36
is the address of a 4-byte area containing the CVDA from RLSACCESS.

FCIS_ADDR37
is the address of a 8-byte area containing the data from DEFINESOURCE.

FCIS_ADDR38
is the address of a 4-byte area containing the CVDA from INSTALLAGENT.

FCIS_ADDR39
is the address of a 8-byte area containing the data from INSTALLUSRID.

FCIS_ADDR40
is the address of a 4-byte area containing the CVDA from CHANGEAGENT.

FCIS_ADDR41 to FCIS_ADDR51
are not used by file control.

FCIS_ADDR52
is the address of a 4-byte area containing the data from JOURNALNUM.

FCIS_ADDR53
is the address of a 4-byte area containing the data from LOADTYPE.

FCIS_ADDR54
is the address of a 4-byte area containing the data from CFDTPOOL.

FCIS_ADDR55
is the address of a 4-byte area containing the data from TABLENAME.

FCIS_ADDR56
is the address of a 4-byte area containing the data from UPDATEMODEL.

FCIS_ADDR57
is the address of a 4-byte area containing the data from REMOTETABLE.

FCIS_ADDR58
is the address of a 4-byte area containing the CVDA from RBATYPE.

FCIS_ADDR59
is the address of a 8-byte area containing the data from CHANGEUSRID.

FCIS_ADDR60
is the address of a 4-byte area containing the data from CHANGEAGREL.

FCIS_ADDR61
is the address of a 8-byte area containing the data from DEFINETIME.

FCIS_ADDR62
is the address of a 8-byte area containing the data from CHANGETIME.

FCIS_ADDR63
is the address of a 8-byte area containing the data from INSTALLTIME.

Modifying fields in the command-level parameter structure

Some fields that are passed to a file control SPI request are used as input to the request, and some are used as output to the request. The method that your user exit program uses to modify a field depends on the usage of the field.

As a general rule:

- On INQUIRE FILE requests, all fields except FILE are output fields.
- On SET FILE requests, all fields are input fields.

For a full description of the parameters to INQUIRE FILE, see Table 7. For a full description of the parameters to SET FILE, see Table 8 on page 130.

Table 7. INQUIRE FILE requests. The relationship between arguments, keywords, data types, and input/output types.

Argument	Keyword	Data Type	Input/Output
Arg1	FILE	CHAR(8)	See note.
Arg2	DSNAME	CHAR(44)	Output
Arg3	FWDRECSTATUS	BIN(31)	Output

Table 7. INQUIRE FILE requests (continued). The relationship between arguments, keywords, data types, and input/output types.

Argument	Keyword	Data Type	Input/Output
Arg4	STRINGS	BIN(31)	Output
Arg5	BASEDSNAME	CHAR(44)	Output
Arg6	LSRPOOLNUM	BIN(31)	Output
Arg7	READ	BIN(31)	Output
Arg8	UPDATE	BIN(31)	Output
Arg9	BROWSE	BIN(31)	Output
Arg10	ADD	BIN(31)	Output
Arg11	DELETE	BIN(31)	Output
Arg12	DISPOSITION	BIN(31)	Output
Arg13	EMPTYSTATUS	BIN(31)	Output
Arg14	OPENSTATUS	BIN(31)	Output
Arg15	ENABLESTATUS	BIN(31)	Output
Arg16	RECOVSTATUS	BIN(31)	Output
Arg17	ACCESSMETHOD	BIN(31)	Output
Arg18	TYPE	BIN(31)	Output
Arg19	OBJECT	BIN(31)	Output
Arg20	REMOTESYSTEM	CHAR(4)	Output
Arg21	REMOTENAME	CHAR(8)	Output
Arg22	RECORDFORMAT	BIN(31)	Output
Arg23	BLOCKFORMAT	BIN(31)	Output
Arg24	KEYLENGTH	BIN(31)	Output
Arg25	KEYPOSITION	BIN(31)	Output
Arg26	RECORDSIZE	BIN(31)	Output
Arg27	RELTYPE	BIN(31)	Output
Arg28	EXCLUSIVE	BIN(31)	Output
Arg29	BLOCKKEYLEN	BIN(31)	Output
Arg30	BLOCKSIZE	BIN(31)	Output
Arg31	*	*	*
Arg32	BUSY	BIN(31)	Output
Arg33	TABLE	BIN(31)	Output
Arg34	MAXNUMRECS	BIN(31)	Output
Arg35	READINTEG	BIN(31)	Output
Arg36	RLSACCESS	BIN(31)	Output
Arg37	DEFINESOURCE	CHAR(8)	Output
Arg38	INSTALLAGENT	BIN(31)	Output
Arg39	INSTALLUSRID	CHAR(8)	Output
Arg40	CHANGEAGENT	BIN(31)	Output
Arg41 to Arg51	*	*	*

Table 7. INQUIRE FILE requests (continued). The relationship between arguments, keywords, data types, and input/output types.

Argument	Keyword	Data Type	Input/Output
Arg52	JOURNALNUM	BIN(15)	Output
Arg53	LOADTYPE	BIN(31)	Output
Arg54	CEDTPOOL	CHAR(8)	Output
Arg55	TABLENAME	CHAR(8)	Output
Arg56	UPDATEMODEL	BIN(31)	Output
Arg57	REMOTETABLE	BIN(31)	Output
Arg58	RBATYPE	BIN(31)	Output
Arg59	CHANGEUSRID	CHAR(8)	Output
Arg60	CHANGEAGREL	BIN(31)	Output
Arg61	DEFINETIME	CHAR(8)	Output
Arg62	CHANGETIME	CHAR(8)	Output
Arg63	INSTALLTIME	CHAR(8)	Output

Note: The file parameter on INQUIRE FILE commands is:

- An input field if the request does not specify START, NEXT, or END
- An output field if the request specifies NEXT
- Omitted if the request specifies START or END.

Table 8. SET FILE requests. The relationship between arguments, keywords, data types, and input/output types.

Argument	Keyword	Data Type	Input/Output
Arg1	FILE	CHAR(8)	Input
Arg2	DSNAME	CHAR(44)	Input
Arg3	FWDRECSTATUS	BIN(31)	Input
Arg4	STRINGS	BIN(31)	Input
Arg5	*	*	*
Arg6	LSRPOOLNUM	BIN(31)	Input
Arg7	READ	BIN(31)	Input
Arg8	UPDATE	BIN(31)	Input
Arg9	BROWSE	BIN(31)	Input
Arg10	ADD	BIN(31)	Input
Arg11	DELETE	BIN(31)	Input
Arg12	DISPOSITION	BIN(31)	Input
Arg13	EMPTYSTATUS	BIN(31)	Input
Arg14	OPENSTATUS	BIN(31)	Input
Arg15	ENABLESTATUS	BIN(31)	Input
Arg16	RECOVSTATUS	BIN(31)	Input
Arg17	*	*	*
Arg18	*	*	*
Arg19	*	*	*

Table 8. SET FILE requests (continued). The relationship between arguments, keywords, data types, and input/output types.

Argument	Keyword	Data Type	Input/Output
Arg20	*	*	*
Arg21	*	*	*
Arg22	*	*	*
Arg23	*	*	*
Arg24	*	*	*
Arg25	*	*	*
Arg26	*	*	*
Arg27	*	*	*
Arg28	EXCLUSIVE	BIN(31)	Input
Arg29	*	*	*
Arg30	*	*	*
Arg31	*	*	*
Arg32	*	*	*
Arg33	TABLE	BIN(31)	Input
Arg34	MAXNUMRECS	BIN(31)	Input
Arg35	READINTEG	BIN(31)	Input
Arg36	RLSACCESS	BIN(31)	Input
Arg37	*	*	*
Arg38	*	*	*
Arg39	*	*	*
Arg40	*	*	*
Arg58	*	*	*
Arg59	*	*	*
Arg60	*	*	*
Arg61	*	*	*
Arg62	*	*	*
Arg63	*	*	*

Modifying input fields:

The correct method of modifying an input field is to create a new copy of it, and to change the address in the command-level parameter list to point to your new data.

Do not modify an input field by altering the data that is pointed to by the command-level parameter list. To do so would corrupt storage belonging to the application program and would cause a failure when the program attempted to reuse the field.

Modifying output fields: The technique described in "Modifying input fields" is not suitable for modifying output fields. (The results would be returned to the new area instead of the application's area, and would be invisible to the application.)

An output field is modified by altering the data that is pointed to by the command-level parameter list. In the case of an output field, you can modify the application's data in place, because the application is expecting the field to be modified anyway.

Modifying the EID

It is not possible to modify the EID to make major changes to requests. It is not possible, for example, to change an INQUIRE FILE request to a SET FILE request. However, you can make minor changes to requests, such as to turn on the existence bit for a variable that had not been specified on the current request.

The following paragraph lists the bits in the EID that can be modified. Any attempt to modify any other part of the EID is ignored.

Your exit program can modify any bit in FCIS_BITS1, FCIS_BITS2, FCIS_BITS3, FCIS_BITS4, FCIS_BITS5, FCIS_BITS6 and FCIS_BITS7, *except for*:

- The existence bit for the FILE keyword.
- The bits for the START, NEXT, END, DEFINESOURCE, INSTALLAGENT, INSTALLUSRID and CHANGEAGENT keywords.
- Any bits described as "not used by file control".
- Any bit corresponding to a keyword that is not applicable to the command being executed. For example, the bit for the CLOSED keyword can be modified on a SET FILE request but not on an INQUIRE FILE request, because CLOSED has meaning only for a SET FILE request. See the descriptions in Table 7 on page 128 and Table 8 on page 130.

Your program can provide its own command-level parameter structure and EID, in which case you should modify UEPCLPS and TS_ADDR0 respectively to point to the new structures.

The EID is reset to its original value before return to the application program. That is, changes to the EID are retained for the duration of the file control SPI request only.

Note: If you modify the EID, you must be careful not to create inconsistent parameters. For example, if the original request specified SET FILE OPEN and your exit turned on the EID bit for CLOSED, the resulting SET FILE request would specify both OPEN and CLOSED. In this case, the results of the command would be unpredictable.

Modifying user arguments

The way in which a user exit program can modify a user argument depends on whether the argument is an input or an output.

- For input arguments, your exit program should obtain sufficient storage to hold the modified argument, set up the required value, and set the associated pointer in the parameter list to the address of the newly acquired area.
- For output and input/output arguments, your exit program can update the argument in place, because the area of storage is represented in the application by a variable that is expected to receive a value from CICS.

Adding user arguments:

Your exit program can add user arguments, provided that it is allowed to modify the corresponding existence bit in the EID.

Assuming that the argument to be added does not already exist, your exit program must:

1. Obtain storage for the argument to be added
2. Initialize the storage to the required value
3. Select and set up the appropriate pointer from the parameter list
4. Select and set up the appropriate existence bit in Arg0
5. If necessary, modify the parameter list to reflect the new end-of-list indicator.

Removing user arguments:

Your exit program can remove user arguments, provided that it is allowed to modify the corresponding existence bit in the EID.

Assuming that the argument to be removed exists, your exit program must:

1. Switch the corresponding argument existence bit in Arg0 to zero
2. Modify the parameter list to reflect the new end-of-list indicator.

File control file state program exits XFCSREQ and XFCSREQC

Two user exits are provided in the file control state program that you can call before and after a file request.

XFCSREQ
: This exit is called before a file ENABLE, DISABLE, OPEN, CLOSE, or CANCEL CLOSE request is acted on. You can use XFCSREQ to gather information about the state of the file; for example, which file requests (SERVREQs) are valid and which journaling options are set. Based on this information, you can suppress the request, if appropriate. See return code UERCBYP for details.

XFCSREQC
: This exit is called after the file request has been acted on. You can use XFCSREQC to gather information about the data set associated with the file; for example, which recovery options are set. XFCSREQC is invoked even if you have used XFCSREQ to suppress the file request.

For ENABLE, DISABLE, OPEN, and CANCEL CLOSE requests, each exit is invoked only once. However, for CLOSE requests, because a file can be quiesced before actual closure, the exits might be invoked more than once. There are two occasions when the user exits XFCSREQ and XFCSREQC are not invoked during a close request:

1. On a controlled, non-immediate shutdown of CICS, when CICS closes all files.
2. After loading a user maintained data table. When the data table load has completed the source data set is no longer required. CICS subsequently closes and de-allocates the file, leaving the data table open.

A single CLOSE request

For a single CLOSE request, XFCSREQ and XFCSREQC are invoked more than once if closure is attempted while the file is being accessed by other tasks. For example, the result of a CLOSE NOWAIT command in these circumstances is that XFCSREQ is invoked before the closure is attempted. Because there are still users of the file, the closure is delayed. However, because it specified NOWAIT, the CLOSE request completes, and invokes XFCSREQC with UEPFSRSP set to 'UEFSPEND', meaning closure is pending. When all activity against the file is

complete, the file is closed, and XFCSREQ and XFCSREQC are invoked under the task that closed it.

A CLOSE WAIT request

For a CLOSE WAIT request, the exits are invoked as follows. XFCSREQ is invoked, the task requests a closure of the file and waits for the closure to happen. When all activity against the file is complete, the file is closed, and XFCSREQ and XFCSREQC are invoked under the task that closed it. Finally, because the closure has now been completed, the task that issued the CLOSE WAIT is resumed, completes its CLOSE request, and invokes XFCSREQC.

A CANCEL CLOSE request

A CANCEL CLOSE request is issued by CICS in response to an UNQUIESCE command that cancels a pending QUIESCE command. A QUIESCE data set command immediately sets all files opened against the specified data set as unenabled, to prevent new tasks being allowed access to the data set. The close part of the operation, however, waits until the last user task finishes before a file is closed. (This is the same as any close operation against a file.) An UNQUIESCE issued while the close is still waiting causes a CANCEL CLOSE request and the invocation of the XFCSREQ and XFCSREQC exits. Note that a CANCEL CLOSE is issued only for close requests that were initiated by a QUIESCE command, not for any other close requests.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XFCSREQ

This exit is invoked before a file ENABLE, DISABLE, OPEN, CLOSE, or CANCEL CLOSE is attempted.

When invoked

Before a file ENABLE, DISABLE, OPEN, CLOSE, or CANCEL CLOSE is attempted.

Note: For function shipped requests, the exit is invoked on the system where the file is local.

Exit-specific parameters

UEPFSREQ

Address of a 2-byte field that indicates the type of file request. The first byte contains one of the following values:

UEPFSOPN

Open request

UEPFSCLS

Close request

UEPFSENB
Enable request

UEPFSDIS
Disable request

UEPFSCAN
Cancel close file request.

If the first byte indicates an open request (UEPFSOPN), the second byte shows the type of open:

UEPFSNOP
Normal open

UEPFSOFB
Open for backout.

If the first byte indicates a close request (UEPFSCLS), the second byte shows the type of close:

UEPFSNC
Normal close

UEPFSCP
Close pending

UEPFSELM
End of load mode close

UEPFSIMM
Immediate close

UEPFSICP
Immediate close pending

UEPFSQU
RLS quiesce close.

UEPFILE
Address of the 8-byte file name.

UEPFINFO
Address of a storage area containing information about the file. The area can be mapped using the DSECT DFHUEFDS, which contains the following fields:

UEFLNAME
The 8-character file name.

UEDSNAME
The 44-character dsname of the data set associated with the file, if this has been set before the file request was issued.

UEFSERV
One byte indicating the SERVREQ settings for this file. The possible values are:

UEFRDIM
Read valid

UEFUPDIM
Update valid

UEFADDIM
Add valid

UEFDELIM
Delete valid

UEFBRZIM
Browse valid.

UEFDSJL
One byte indicating the automatic journaling options set for this file. The possible values are:

UEFJRO
Journal read-only

UEFJRU
Journal read for update

UEFJWU
Journal write update

UEFJWA
Journal write add

UEFJDSN
Dsname has been journaled

UEFJSYN
Journal read synchronously

UEFJASY
Journal write asynchronously.

UEFDSVJL
One byte indicating a further automatic journaling option which applies to VSAM files only. The value is:

UEFJWAC
Write add complete.

UEFDSJID
One byte containing the number of the journal to be used for automatic journaling, if any.

UEFDSACC
One byte indicating the access method of the file. The possible values are:

UEFVSAM
VSAM file

UEFBDAM
BDAM file

UEFCFDT
Coupling facility data table

UEFBCRV
Set to nulls for this exit.

UEFFRLOG
Set to nulls for this exit.

UEFFRCLG
Set to blanks for this exit.

UEFCDATE
Set to nulls for this exit.

UEFCTIME
Set to nulls for this exit.

UEFBCAS
Set to nulls for this exit.

UEFACBCP
This field is set to nulls in this exit.

Note: Only the first seven fields of UEPFINFO are set for this exit. Of the remaining fields, URFFRCLG is set to blanks, and the others are set to nulls.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

Return codes

UERCNORM
Continue processing.

UERCBYP
Suppress the file request. You cannot use UERCBYP:

- To suppress a CLOSE request if the second byte of UEPFSREQ indicates it is one of the following types of close:
  - End of load-mode close (UEPFSELM)
  - Immediate close (UEPFSIMM)
  - Immediate close pending (UEPFSICP)
- To suppress an OPEN request if a file is being opened to carry out backout processing, because this would cause a backout failure. The second byte of UEPFSREQ is set to UEPFSOFB if the file is being opened for backout.

In the case of a valid suppression, CICS issues message DFHFC0996:

```
Open/Close/Enable/Disable/Cancel of close of file
filename suppressed due to intervention of user exit
```

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

API and SPI calls
All except EXEC CICS SHUTDOWN and EXEC CICS XCTL can be used.

Note:

1. Take care when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a file control request is issued from the XFCSREQ exit. Use of the recursion counter UEPRECUR is recommended.
2. Exit programs that issue EXEC CICS commands must first address the EIB. See "Using CICS services" on page 4.
3. Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See "Returning values to CICS" on page 10.

4. Exit programs can invoke EXEC CICS SET commands against the file whose state change has led to the exit being invoked. However, dependent upon other concurrent activity within the CICS system, there is the potential for a deadlock to occur between tasks that are manipulating the state of the file by means of such SPI commands.

Exit XFCSREQC

This exit is called after a file ENABLE, DISABLE, OPEN, CLOSE, or CANCEL CLOSE command completes.

When called

After a file ENABLE, DISABLE, OPEN, CLOSE, or CANCEL CLOSE command completes.

Note: For function-shipped requests, the exit is called on the system where the file is local.

Exit-specific parameters

UEPFSREQ

Address of a 2-byte field that indicates the type of file request. The first byte contains one of the following values:

UEPFSOPN
Open request

UEPFSCLS
Close request

UEPFSENB
Enable request

UEPFSDIS
Disable request

UEPFSCAN
Cancel file close request.

If the first byte indicates an open request (UEPFSOPN), the second byte shows the type of open:

UEPFSNOP
Normal open

UEPFSOFB
Open for backout.

If the first byte indicates a close request (UEPFSCLS), the second byte shows the type of close:

UEPFSNC
Normal close

UEPFSCP
Close pending

UEPFSELM
End of load mode close

UEPFSIMM
Immediate close

UEPFSICP
Immediate close pending

UEPFSQU
RLS quiesce close.

UEPFILE
Address of the 8-byte file name.

UEPFINFO
Address of a storage area containing information about the file. The area can be mapped using the DSECT DFHUEFDS, which contains the following fields:

UEFLNAME
The 8-character file name.

UEDSNAME
The 44-character dsname of the data set associated with the file.

UEFSERV
One byte indicating the SERVREQ settings for this file. The possible values are:

UEFRDIM
Read valid

UEFUPDIM
Update valid

UEFADDIM
Add valid

UEFDELIM
Delete valid

UEFBRZIM
Browse valid.

UEFDSJL
One byte indicating the automatic journaling options set for this file. The possible values are:

UEFJRO
Journal read-only

UEFJRU
Journal read for update

UEFJWU
Journal write update

UEFJWA
Journal write add

UEFJDSN
Dsname has been journaled

UEFJSYN
Journal read synchronously

UEFJASY
Journal write asynchronously.

UEFDSVJL
One byte indicating a further automatic journaling option which applies to VSAM files only. The value is:

UEFJWAC
Write add complete.

UEFDSJID
One byte containing the number of the journal to be used for automatic journaling, if any.

UEFDSACC
One byte indicating the access method of the file. The possible values are:

UEFVSAM
VSAM file

UEFBDAM
BDAM file

UEFCFDT
Coupling facility data table

UEFBCRV
One byte indicating the recovery attributes of the data set associated with this file. The possible values are:

UEFBCFR
Forward recovery specified

UEFBCLOG
Logging specified

UEFBCVAL
Flag indicating that recovery attributes are valid.

UEFFRLOG
A 1-byte field containing the forward recovery log identifier in the range 1—99, taken from the recovery attributes in the CICS file resource definition. This number corresponds to a CICS internal journal name of the form DFHJ*nn*, where *nn* is the forward recovery log number. CICS maps this journal name to a forward recovery log stream.

The field is set to zero if forward recovery logging is not specified for the file, or if the forward recovery log stream name has been obtained from the ICF catalog.

UEFFRCLG
A 26-byte field containing the name of the forward recovery log stream taken from the ICF catalog, to be used for forward recovery. Set to blanks if not specified in the ICF catalog or if forward recovery is not being used for the file.

UEFCDATE
A date (YYYYDDD+) in packed decimal format. This field is set only when the file is the last file to close against the VSAM sphere with which it is associated. It contains the date when activity against the VSAM sphere was brought to an end (quiesced).

UEFCTIME
A time (HHMMSST+) in packed decimal format. This field is set only when the file is the last file to close against the

VSAM sphere with which it is associated. It contains the time when activity against the VSAM sphere was brought to an end.

UEFBCAS
A flag-byte indicating the availability of this data set. If set, the value is:

UEPFBCAS
Data set marked unavailable.

UEFACBCP
Address of a read-only copy of the ACB for a VSAM file, or the DCB for a BDAM file. Set only after completion of a successful open.

UEPFSRSP
Address of a byte containing the return codes for the request. This has one of the following values:

UEFSNORM
Normal response.

UEFSWARN
Warning response.

UEFSFAIL
Failure response.

UEFSPEND
Pending response. The 'Pending' response can be returned only after a CLOSE request. It indicates that, as a result of the CLOSE request, a closure is pending on the file, the file is being quiesced. When all activity against the file has completed, it is closed. Note that, if enabled, the XFCSREQ and XFCSREQC exits are driven again, when the actual closure takes place.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first called, and is incremented for each recursive call.

Note:

1. The first seven fields of UEPFINFO (UEFLNAME through UEFDSACC) are set for all requests; that is, following an OPEN, CLOSE, ENABLE, or DISABLE request.
2. The next three fields (UEFBCRV, UEFFRLOG, and UEFFRCLG) are valid only after a successful OPEN request.
3. The fields UEFCDATE through UEFCBCAS are set only after a successful CLOSE request. After all other requests, if the file is already closed, if the closure fails, or if the closure is pending, these fields are set to nulls.
4. Exit programs can call EXEC CICS SET commands against the file whose state change leads to the exit being called. However, dependent upon other concurrent activity within the CICS system, there is the potential for a deadlock to occur between tasks that are manipulating the state of the file with such SPI commands.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

API and SPI calls
All except EXEC CICS SHUTDOWN and EXEC CICS XCTL can be used.

Note:

1. Take care when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a file control request is issued from the XFCSREQC exit. Use of the recursion counter UEPRECUR is recommended.
2. Exit programs that issue EXEC CICS commands must first address the EIB. See "Using CICS services" on page 4.
3. Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See "Returning values to CICS" on page 10.

The DFH$REQC sample global user exit program:

DFH$REQC provides sample processing for the file control state program global user exit, XFCSREQC. The exit program, if enabled at the XFCSREQC exit point, is invoked after a file ENABLE, DISABLE, OPEN, CLOSE, or CANCEL CLOSE request has been acted on.

There is more information about using the sample program in the comments in the DFH$REQC source code. In summary, DFH$REQC performs the following processing :

1. Checks whether an open request for a VSAM data set has been acted on and that the ACB error flag (ACBERFLG) is non zero.
2. If all are true, performs default processing. The ACBERFLG error code is checked. If it is equal to X'74', the following message and data areas are built into 690 bytes of contiguous storage, in such a way that they conform to the editing rules of the WRITE OPERATOR command:

   Message

   ```
   VSAM Open error has occurred – VSAM error flag X'nn'
   ```

   The ACBERFLG error code is converted from hex to character format and appended to the end of the message.

   Access method control block (ACB)
   The data starts with an 'ACB' eye-catcher. All the data within the ACB is converted into character format and added after the eye-catcher.

   Exec Interface Block (EIB)
   The data starts with an 'EIB' eye-catcher. All the data within the EIB is converted into character format and added after the eye-catcher.

   Parameter list passed to the exit program
   The data starts with a 'PLIST' eye-catcher. Only the exit-specific parameters are converted to character format and added after the eye-catcher.

File Information
The data starts with a 'FINFO' eye-catcher. This is followed by the 8-character file name and the 44-character dsname of the data set associated with the file, as described by fields UEFLNAME and UEDSNAME in the DFHUEFDS DSECT.

3. Issues a WRITE OPERATOR command to write as much data as has been created in the 690 bytes of storage to the system console.
4. Returns a normal response of zero in register 15.

The sample program contains logic to write the same messages and data areas that are written on the default processing path to a transient data queue. In this case, output is edited into lines of 132 bytes. Each line is written when it is full, at which point (or upon request) a new data area or message is started.

In addition, the program logic allows for other VSAM error codes to be tested. These can be actioned to follow existing paths within the program, or tailored to use all or part of the set of messages and data areas already defined. Alternatively, the sample program can be customized to expand the set of messages and data areas written.

Related concepts:

"File control state sample exit program (DFH$REQC)" on page 24
CICS includes a sample global user exit program for use with the file control state program. The sample program is DFH$REQC.

File control open/close program exit XFCNREC

You can use XFCNREC to suppress the open failure on a non-RLS data set.

For RLS data sets, recovery is a property of the data set. Therefore it is not possible for files and their base data set to have unmatched recovery attributes. For more information about writing an XFCNREC exit program, see the *CICS Recovery and Restart Guide*.

Exit XFCNREC

When invoked

Before file open, when a mismatch is detected:

1. Between the backout recovery setting for the file and its associated non-RLS data set.
2. Because BWO is required but the recovery attributes indicate that no associated forward recovery file has been specified.

Exit-specific parameters

UEFILE
Address of the 8-byte file name. If the file name is less than 8 characters in length, it will be padded with blanks.

UEDSETN
Address of the 44-byte base data set name. If the data set name is less than 44 characters in length, it will be padded with blanks.

UEPFRCV
Address of a 1-byte field containing the backout recovery setting for the file, as specified in the FILE definition. The possible value is:

UEPFLOG
Backout logging specified.

If RECOV(NONE) is specified in the FILE definition, the addressed field contains hexadecimal zeros.

This field has no meaning if the exit is driven because of a BWO mismatch.

UEPFAIL
Address of a 1-byte field containing the reason for the mismatch. The possible values are:

UEPBWOF
A BWO mismatch

UEPATTF
A mismatch in the backout recovery settings

UEPOPEN
Address of a 1-byte field. Its default value is N. To bypass an open failure caused by a BWO mismatch, set the addressed field to Y.

Return codes

UERCNORM
Fail open as normal.

UERCBYP
Bypass open failure—accept mismatch.

XPI calls
Must not be used.

SPI calls
Must not be used.

API and SPI calls
Must not be used.

XFCNREC exit with a backout recovery setting mismatch

Use the XFCNREC global user exit if you want to continue with open processing, even though the backout recovery settings for different files associated with the same base data set are not consistent.

After an open failure has been suppressed, CICS can no longer guarantee integrity for the data set and marks it accordingly. If you use the **EXEC CICS INQUIRE DSNAME** or **CEMT INQUIRE DSNAME RECOVSTATUS** commands after an open failure has been suppressed, a response of NOTRECOVABLE is returned. Logging continues for the data set for requests. Logging uses only files that have BACKOUT defined.

The mismatched state of the data set continues until an **EXEC CICS** or **CEMT SET DSNAME REMOVE** command is issued, or until an initial or cold start of CICS, if the associated data set is not in a backout failed state.

At the point when the mismatch is accepted, CICS issues a message to warn that integrity can no longer be guaranteed. The order in which files are opened for the same base data set determines the content of the message that is received.

If the base cluster block is set as unrecoverable and a mismatch has occurred, access is granted to the data set with an unrecoverable file, before the data set is fully recovered.

Three parameters are passed to the XFCNREC exit to provide a means of selecting which mismatches to accept and which to reject. These parameters are the address of the file name, the address of the base data set name, and the address of a byte containing the file backout indicator. Because the exit is driven only if there is a mismatch, the data set backout indicator can be derived from the setting for the file.

Note: If XFCNREC is used to suppress an open failure due to a mismatch, the global user exit XFCSREQC passes the base data set backout setting as the exit parameter UEFBCRV, and not the file backout setting, which might be different.

Using XFCNREC with a BWO mismatch

Exit XFCNREC can allow the file to be opened and CICS will continue to run normally. However, forward recovery will not be available for the opened data set.

File control quiesce receive exit, XFCVSDS

The XFCVSDS exit is invoked when VSAM RLS notifies CICS that processing is required as a result of some data set-related events occurring in the sysplex.

XFCVSDS is invoked before CICS processing takes place, and only if a data set name block (DSNB) exists for the data set. The actions that cause XFCVSDS to be invoked are:

- **A data set is being quiesced throughout the sysplex**.

  CICS is informed about this action only if it has files open in RLS mode against the data set.

  If CICS is notified about a quiesce action, the XFCVSDS global user exit program can cancel the data set quiesce, in which case it cancels the quiesce throughout the sysplex, and the data set remains in the unquiesced state.
- **A data set is being unquiesced throughout the sysplex**.

  All CICS regions in the sysplex that are registered with a VSAM RLS control ACB are informed about unquiesce actions.
- **DFSMSdss wants to start a non-BWO backup of a data set**.

  CICS is notified about a non-BWO backup start action only if it has files open in RLS mode against the data set.

  If CICS is notified about a non-BWO backup start action, XFCVSDS can be used to cancel the backup.
- **DFSMS has completed a non-BWO backup of a data set**.

  All CICS regions in the sysplex that are registered with a VSAM RLS control ACB are informed about non-BWO backup complete actions.
- **DFSMS wants to start a BWO backup of a data set**.

  CICS is notified about a BWO backup start action only if it has files open in RLS mode against the data set.

  If CICS is notified about a BWO backup start action, XFCVSDS can be used to cancel the backup.
- **DFSMS has completed a BWO backup of a data set**.

  All CICS regions in the sysplex that are registered with a VSAM RLS control ACB are informed about BWO backup complete actions.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully

reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XFCVSDS

Exit XFCVSDS is invoked after VSAM RLS has informed CICS that processing is required as a result of a data set-related action occurring in the sysplex.

When invoked

Invoked after VSAM RLS has informed CICS that processing is required as a result of a data set-related action occurring in the sysplex.

Exit-specific parameters

UEPDSNAM
Address of a 44-byte field containing the name of the data set to which the action applies

UEPVSACT
Address of a 1-byte field indicating the RLS action of which CICS has been informed. Possible values are:

UEQUIES
Quiesce

UEUNQUIS
Unquiesce

UENBWST
Non-BWO backup start

UENBWCMP
Non-BWO backup complete

UEBWOST
BWO backup start

UEBWOCMP
BWO backup complete.

UEPQUCLS
Address of a 1-byte field indicating, for UEQUIES only, how files open in RLS mode are to be closed. Possible values are:

UEORDCLO
Wait until all in-flight UOWs that are accessing the data set have completed syncpoint before closing.

UEIMMCLO
Abend all in-flight UOWs that are accessing the data set before closing.

UEPCPTEC
Address of a 1-byte field indicating, for UENBWST and UEBWOST only, whether the backup is to use the concurrent copy technique. Possible values are:

UEORDCOP
Concurrent copy is not being used.

UECONCOP
Concurrent copy is being used.

Return codes

UERCNORM
Continue processing—complete the actions required to support the VSAM RLS operation.

UERCBYP
This applies only to actions UEQUIES, UENBWST and UEBWOST. CICS is *not* to carry out the processing required for the VSAM RLS action, and is to cancel the action throughout the sysplex.

A return code of UERCPURG is not allowed.

XPI calls
All can be used.

API and SPI calls
You can use CICS API and SPI commands at this exit. In general all can be used, with the following restrictions:

- You should avoid the use of commands that cause the issuing task to suspend.
- You must not use EXEC CICS SHUTDOWN or EXEC CICS XCTL.
- You must not use the QUIESCESTATE option of EXEC CICS SET DSNAME for data set UEPDSNAM.
- You must not use the OPENSTATUS option of EXEC CICS SET FILE, or issue file control requests, for files that reference data set UEPDSNAM.

File control quiesce send exit XFCQUIS

The XFCQUIS global user exit is invoked on completion of a VSAM RLS quiesce or unquiesce of a data set that was requested either by a CEMT or **EXEC CICS SET DSNAME QUIESCESTATE** command.

The exit is invoked regardless of whether the QUIESCESTATE action has completed successfully or unsuccessfully. This enables you to perform, or schedule, any processing that cannot take place until quiesce or unquiesce processing has finished.

When invoked
On completion, successful or failed, of a SET DSNAME QUIESCESTATE command.

Exit-specific parameters

UEPQDSNM
Address of a 44-byte field containing the name of the data set that was being quiesced or unquiesced.

UEPQSTAT
Address of a 1-byte field indicating whether the data set was being quiesced or unquiesced. Possible values are:

UEQSD
Data set was being quiesced by

QUIESCESTATE(QUIESCED). In-flight UOWs accessing the data set completed syncpoint before files open in RLS mode were closed.

UEIMQSD
Data set was being quiesced by QUIESCESTATE(IMMQUIESCED). In-flight UOWs accessing the data set were abended before files open in RLS mode were closed.

UEUNQSD
Data set was being unquiesced by QUIESCESTATE(UNQUIESCED).

UEPQRCDE
Address of a 1-byte field indicating the result of the quiesce or unquiesce. Possible values are:

UEQOK
Successful.

UEQREJEC
Rejected—see UEPQCONF for the reason code.

UEQUNKNO
Failed because data set not known to DFSMS as a VSAM data set.

UEQIOERR
Failed because of RLS error or SMSVSAM server not available.

UEQCANCL
Failed because quiesce was canceled by user (UEQSD and UEIMQSD only).

UEQTIMED
Failed because quiesce was canceled due to timeout (UEQSD and UEIMQSD only).

UEQMIGRT
Failed because the data set has been migrated.

UEPQCONF
Address of a 1-byte field indicating the reason why the quiesce or unquiesce was rejected (for UEQREJEC only). Possible values are:

UEQUIINP
Quiesce is in progress (UEQSD and UEIMQSD status only).

UEUNQINP
Unquiesce is in progress.

UENBWINP
Non-BWO copy is in progress.

UEBWOINP
BWO copy is in progress.

UEUNKINP
Unknown event is in progress.

Return codes

UERCNORM
Continue processing.

A return code of UERCPURG is not allowed.

XPI calls
All can be used.

API and SPI calls
You can use CICS API and SPI commands at this exit. In general, all except EXEC CICS SHUTDOWN and EXEC CICS XCTL can be used, but you must not use the QUIESCESTATE keyword of EXEC CICS SET DSNAME.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

File control recovery program exits XFCBFAIL, XFCBOUT, XFCBOVER, and XFCLDEL

CICS provides four global user exits that you can use in connection with file control recovery operations.

These are:

XFCBFAIL
Invoked when an error occurs during backout.

XFCBOUT
Invoked when CICS is about to back out a file update.

XFCBOVER
Invoked when CICS is about to skip unit-of-work (UOW) backout because a batch program has overridden RLS retained lock protection and opened a data set for batch processing.

XFCLDEL
Invoked when backing out a write to a BDAM or a VSAM ESDS data set.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Order of invocation

Each of the exits in the file control recovery program might be invoked during an attempt to backout a file update. If the backout fails, each exit might be re-invoked when the backout is retried. If an exit program needs to determine whether it is being invoked during the original backout attempt, or during a retry, it can check the value of the RE_ATTACHED_TRANSACTION field returned by an XPI INQUIRE_TRANSACTION call.

The way in which the exits interact, and the order in which they are invoked, is shown in the following list. Assuming that all the exits are enabled, for each backout attempt or retried backout attempt:

1. If an open during backout fails, XFCBFAIL is invoked. **None of the other exits is invoked**.
2. If the SHCDS PERMITNONRLSUPDATE command has been issued for the data set being backed out, XFCBOVER is invoked. **If it returns UERCNORM (do not perform the backout), no further exits are invoked**.
3. Unless item 1 applies, or XFCBOVER has been invoked and returned UERCNORM, XFCBOUT is invoked.
4. Backout issues a read update request for the record being backed out.

   If the read update fails, XFCBFAIL is invoked, followed by **no further exits**.
5. If the update to be backed out was a write to a data set which does not support physical deletes (that is, a BDAM data set or a VSAM ESDS), XFCLDEL is invoked.
6. If a failure occurs after this point, XFCBFAIL is invoked.

Enabling the exit programs

To enable an exit program you must complete one of two possible actions.

To enable these exits, you must do one of the following:

- Specify the system initialization parameter TBEXITS=(name1,name2,name3,name4,name5,name6), where name1 through name6 are the names of your user exit programs for XRCINIT, XRCINPT, XFCBFAIL, XFCLDEL, XFCBOVER, and XFCBOUT.
- Enable the exits during the first stage of initialization using a PLTPI program.

If you use the TBEXITS parameter to enable the exits, a global work area of 4 bytes is provided. If you use a PLTPI program, you can select the size of the global work area. You can also enable more than one exit program for use at each exit point; the TBEXITS parameter allows only one exit program at each exit point. PLTPI processing is described in Chapter 5, "Writing initialization and shutdown programs," on page 409.

Exit XFCBFAIL, file control backout failure exit

Exit XFCBFAIL is invoked whenever there is a failure during backout of an update made to a file record.

If, within a given UOW, there are backout failures for more than one record in the same file, or for records in multiple files, the exit is invoked:

- For the first record in each data set for which backout fails.

  If more than one file is associated with a single data set, only the first record in the first of the files to fail backout within UOW causes CICS to invoke the exit. All subsequent records are failed with the same error, but the exit is not invoked again.
- For the first record for each data set that fails during any retry of the backout for this UOW.

It is not invoked for backout failures to other (non-file-control) resources within the UOW.

For VSAM data sets, backout failure processing saves information that allows the backout to be retried later.

For BDAM data sets, the backout cannot be retried. If backout fails against a BDAM data set, you can use the XFCBFAIL exit to preserve data integrity by terminating CICS immediately. If the XFCBFAIL exit is not enabled, or does not terminate CICS, the BDAM data is committed and the locks are released. If the exit is enabled, you can use the XFCBFAIL global user exit program to save information that you can use to manually correct the data. However, you need to be careful that in doing this you do not back out other changes made between the time of the backout failure and the time of your own manual recovery action.

When invoked

If an error occurs during backout of a change made to a file (on the first failure in the UOW for the data set associated with the file).

Exit-specific parameters

UEPBLOGR

Address of the file control portion of the log record that represents the update that was being backed out when the file control failure occurred. The log record can be mapped using the DSECT DFHFCLGD.

UEPTRANS

Address of a 4-byte field containing the transaction id under which the update that is being backed out was made.

UEPTRMNL

Address of the 4-byte terminal id for the terminal or principal facility from which the update that is being backed out was made.

UEPTASK

Address of the 4-byte (packed decimal) field containing the task number for the task under which the update that is being backed out was made.

UEPFCRSP

Address of the file control response byte. This can have one of the following values:

UEAIXFUL

No space in non-unique alternate index.

UECACHE

RLS cache failure or cache connectivity failure.

UENBWBAK

Non-BWO backup in progress.

UEDLOCK
Deadlock detected.

UEDUPREC
Duplicate key on unique alternate index.

UEIOEROR
I/O error.

UELCKFUL
RLS lock structure full.

UENOLDEL
Logical delete not carried out (XFCLDEL exit point is either not enabled or the XFCLDEL global user exit program chose not to perform the logical delete).

UENOSPAC
Data set out of storage.

UEOPENER
Error opening the file.

UERLSERR
SMSVSAM RLS server failure.

UERLSDIS
RLS access is currently disabled.

UERLSCON
Attempt to continue a thread with a new instance of the SMSVSAM RLS server.

UEUNEXP
Unexpected error.

UEPERR
Address of a one-byte field containing the error type. The values of the error-byte and their meanings are described in "Values of the error-type byte referenced by UEPERR" on page 153.

Return codes

UERCNORM
Continue processing and invoke CICS backout failure control.

This causes a backout failure error message to be issued (DFHFC4701 for a VSAM data set, and DFHFC4702 for a BDAM data set). For a VSAM data set CICS converts the record lock into a retained lock, and the log record is saved for a later retry of the backout.

UERCBYP
Ignore the error (do not invoke CICS backout failure control) and continue. Setting this return code could be damaging to the integrity of your data.

A return code of UERCPURG is not allowed. There is no need to set a UERCPURG return code, because the conditions under which this exit is invoked mean that a purged condition cannot be returned by any XPI or API calls.

XPI calls
All can be used, but subject to the same caution as for API and SPI calls.

API and SPI calls

Although this exit is allowed to issue API and SPI calls, you should be very careful about which commands you use because the exit is invoked during file backout, which is part of syncpoint phase 2.

It is recommended that you restrict EXEC CICS commands to inquiries, and avoid commands that update CICS resources, because the resources may themselves be in a state of recovery. In particular, the following restrictions apply:

1. Do not issue any recoverable operations.
2. Do not use operations that access systems or resource owners external to this CICS, even if the target resource is non-recoverable.
3. Do not disable or close files, because this could cause further error conditions.
4. It is possible for this exit to be invoked under a different transaction environment from that under which the updates that are being backed out were originally made. If your exit program wants to perform any actions (such as writing a message to the terminal) that require it to be running under the original transaction environment, it must first check the value returned in the RE_ATTACHED_TRANSACTION parameter of a transaction manager INQUIRE_TRANSACTION XPI call.

Values of the error-type byte referenced by UEPERR:

The UEPERR field in the XFCBFAIL parameter list points to an error-type byte, which contains one of several possible values.

Possible values are:

XBFERU

An error response has been returned from the file control file-request-handler program while processing a READ UPDATE request. This request is issued by file control backout to retrieve the existing copy of the record before backing it out.

Use UEPFCRSP in combination with the type of record, shown in the file control portion of the log record addressed by parameter UEPBLOGR, to determine the specific problem. The storage area addressed by UEPBLOGR contains either the before-image of a “read-update” record or the new copy of a “write-add” to be deleted. The type-of-record field, FLJB_RECORD_TYPE, is defined in DSECT DFHFCLGD.

XBFERE

An error response has been returned from the file control file-request-handler program while processing a REWRITE request. This request is issued by file control backout to replace the existing copy of the record on the data set with the “before-image” held in the log record addressed by UEPBLOGR. Use parameter UEPFCRSP to determine which error occurred.

XBFEWR

An error response has been returned from the file control file-request-handler program while processing a WRITE request. This request is issued by file control backout to add the “before-image” of a deleted record. Use parameter UEPFCRSP to determine which error occurred.

XBFEDL
An error response has been returned from the file control file-request-handler program while processing a REWRITE DELETE request. This request is issued by file control backout to delete a new record added to a VSAM data set. Use parameter UEPFCRSP to determine which error occurred.

XBFENO
The failure that occurred during file control backout was not as a result of an error response from the file control file-request-handler program. Use parameter UEPFCRSP to determine which error occurred.

DFH$FCBF sample global user exit program:

DFH$FCBF provides sample processing for the file control backout failure global user exit, XFCBFAIL. The exit program, if enabled at the XFCBFAIL exit point, is invoked if an error occurs during backout of a file control update.

There is more information about using the XFCBFAIL user exit, and the sample program, in the comments within the DFH$FCBF source code. The comments also include some suggested extensions to the sample program.

In summary, DFH$FCBF performs the following processing:

- If tracing is active for file control, makes a user trace entry. This has trace point id X'01D0' and traces:
  - An eye-catcher 'DFH$FCBF ENTRY'
  - The file control response byte
  - The error type
  - The file control portion of the log record.
- Issues an EXEC CICS INQUIRE FILE command to check the access method to see if the data set is BDAM. If it is, CICS does not support backout retries. Therefore, a message is written to the console advising that either this file and any other files using the base data set (named in message DFHFC4702) should be closed, or CICS should be shut down to prevent further corruption. Sets a response of UERCNORM and takes the normal exit from the program.
- If the access method is neither BDAM nor VSAM, takes the error exit from the program.
- Checks whether the file is one for which it has been decided that backout failures will be ignored, by checking the filename (field FLJB_FILE_NAME in the log record). The sample program writes a message to the console to this effect, then sets a response of UERCBYP and takes the normal exit from the program. A return code of UERCBYP specifies that the error should be ignored. This causes CICS not to retry the backout; the result is as if the data were committed instead of being backed out.

  The sample program takes this step to demonstrate the use of the UERCBYP return code. It is not recommended that you use UERCBYP with important data sets.
- Examines the file control response code and issues a message to the console describing the procedure to be followed for this error. The sample program provides slots for each possible file control response code, and includes suggested messages for some of them. The sample program should be customized by expanding the set of messages to describe procedures that are appropriate for your installation for each error, or to take other action within the exit program. If you do not add a message for any particular response code, the

operator still sees message DFHFC4701, which advises on any action that needs to be taken. This is issued as part of CICS backout failure processing.

- Sets a response code of UERCNORM and takes the normal exit from the program. A return code of UERCNORM specifies that CICS backout failure processing is to be carried out. This means that CICS issues a backout failure error message and, for a VSAM data set, ensures that the record remains locked until the backout can be retried and saves the log record for later retry.
- Normal exit from the program writes a user trace entry if tracing is active for file control and there were no errors during processing. This has trace point id X'01D1' and traces:
  - An eye-catcher 'DFH$FCBF EXIT OK'
  - The file control portion of the log record
  - Some text: 'Handle backout failure' or 'Bypass backout failure' as appropriate.
- Error exit from the program (taken if errors occur during processing or if CICS functions fail):
  - Writes a user exception trace entry regardless of the trace setting. This has trace point id X'01D2' and traces:
    - An eye-catcher 'USEREXC'
    - An eye-catcher 'DFH$FCBF EXIT FAIL'
    - The file control portion of the log record.
  - Returns a response of UERCNORM so that, although an error has occurred in the exit, CICS still performs its normal backout failure processing.

Related concepts:

"File control recovery sample exit programs (DFH$FC*)" on page 24
CICS includes three sample file control global user exit programs. The sample programs are DFH$FCBF, DFH$FCBV, DFH$FCLD.

Exit XFCBOUT, file control backout exit

XFCBOUT is invoked when a file control update is about to be backed out. The log record containing the before-image of the record being backed out is passed to the exit program.

XFCBOUT does not provide a return code to allow your exit program to bypass the backout of the update, because this would result in data corruption. However, the file name is in the log record, so your exit program can use an **EXEC CICS INQUIRE FILE** command to get information about the file.

When invoked

Invoked when an update (represented by a before-image log record) is being backed out by File Control.

Exit-specific parameters

UEPFLOGR

The address of the file control portion of the log record that is being presented for backout. This is mapped by the DSECT DFHFCLGD.

Return codes

UERCNORM

Continue processing.

A return code of UERCPURG is not allowed. There is no need to set a UERCPURG return code, because this exit is invoked during syncpoint phase 2, and therefore cannot get a purged response from any calls it makes.

XPI calls
All can be used, but subject to the same caution as for API and SPI calls.

API and SPI calls
Although this exit is allowed to issue API and SPI calls, you should be very careful about which commands you use because the exit is invoked during file backout, which is part of syncpoint phase 2.

It is recommended that you restrict EXEC CICS commands to inquiries, and avoid commands that update CICS resources, because the resources may themselves be in a state of recovery. In particular, the following restrictions apply:

1. Do not issue any recoverable operations.
2. Do not use operations that access systems or resource owners external to this CICS, even if the target resource is unrecoverable.
3. Do not disable or close files, because this could cause further error conditions.
4. It is possible for this exit to be invoked under a different transaction environment from that under which the updates that are being backed out were originally made. If your exit program wants to perform any actions (such as writing a message to the terminal) that require it to be running under the original transaction environment, it must first check the value returned in the RE_ATTACHED_TRANSACTION parameter of a transaction manager INQUIRE_TRANSACTION XPI call.

Because it is anticipated that XFCBOUT will be used for specific applications, no general-purpose sample exit program is provided. You could use any of the samples for the other file control recovery exits, DFH$FCBF, DFH$FCBV, or DFH$FCLD, as the basis for an XFCBOUT exit program.

Exit XFCBOVER, file control backout override exit

Exit XFCBOVER is part of the support CICS file control provides for *batch windows* in a VSAM RLS environment.

VSAM RLS locks individual records within a data set, and these locks are converted to retained locks for those UOWs that are not completed because of backout or indoubt failures, thus preserving data integrity. To avoid corruption of a data set by a non-RLS batch job, which is not aware of the retained record locks, a data set cannot normally be opened for update in non-RLS mode if it has any locked records.

Retained lock override for batch:

There may be circumstances in which you want to override these locks and force the open of a data set for batch processing.

For example, when:

- There is insufficient time available, before running the batch job, in which to resolve the situation that caused the records to be locked, or

- It is known that the batch job cannot harm data integrity (because it does not update existing records in the data set, or it does not update any records that CICS may have updated).

To override the open restriction, VSAM RLS provides the SHCDS PERMITNONRLSUPDATE command, to allow a non-RLS batch job to open a sphere for update even when there are retained locks.

Effect of retained lock override on CICS:

VSAM records the use of the option to override retained locks, so that it can notify a CICS region when the region next opens the data set. Because data could have been altered by the non-RLS batch job, the results of CICS performing any recovery (on UOWs that were in a backout-failed or indoubt-failed state at the time of the batch job) are unpredictable. In this situation, therefore, the default CICS action is not to back out any updates that were outstanding at the time that locks were overridden, and to write diagnostic information about each backout ignored to the CSFL transient data queue.

The XFCBOVER global user exit is provided to enable you, for each UOW log record for which backout is being ignored, to:

- Write application-related diagnostics to supplement those provided by CICS
- To perform application-related recovery actions
- To reverse the default by requesting that the backout should be carried out after all. This option is required for the case where the batch job is known not to corrupt data integrity (for example, because it only inserts records).

When invoked

Whenever CICS is about to ignore a UOW log record that is due to be backed out, because the lock that protected the updated record could have been overridden by a non-RLS batch program.

Exit-specific parameters

UEPOLOGR

Address of the file control portion of a shunted log record that represents an update to a data set for which retained locks may have been overridden. The file control portion of the log record can be mapped using the DSECT DFHFCLGD.

UEPODSN

Address of a 44-byte area of storage containing the name of the data set whose locks were overridden.

Return codes

UERCNORM

Do not perform the backout of this log record. Any updates performed by the batch run should take precedence.

UERCBCKO

Perform the backout. It is known that the actions of the batch job could not have affected this update.

A return code of UERCPURG is not allowed. There is no need to set a UERCPURG return code, because this global user exit is invoked during syncpoint phase 2, and therefore cannot get a purged response from any calls that it makes.

XPI calls

All can be used, but subject to the same caution as for API and SPI calls.

API and SPI calls

Although this exit is allowed to issue API and SPI calls, you should be very careful about which commands you use because the exit is invoked during file backout, which is part of syncpoint phase 2.

It is recommended that you restrict EXEC CICS commands to inquiries, and avoid commands that update CICS resources, because the resources may themselves be in a state of recovery. In particular, the following restrictions apply:

1. Do not issue any recoverable operations.
2. Do not use operations that access systems or resource owners external to this CICS, even if the target resource is non-recoverable.
3. Do not disable or close files, because this could cause further error conditions.
4. It is possible for this exit to be invoked under a different transaction environment from that under which the updates that are being backed out were originally made. If your exit program wants to perform any actions (such as writing a message to the terminal) that require it to be running under the original transaction environment, it must first check the value returned in the RE_ATTACHED_TRANSACTION parameter of a transaction manager INQUIRE_TRANSACTION XPI call.

DFH$FCBV sample global user exit program:

DFH$FCBV provides sample processing for the file control backout override global user exit, XFCBOVER.

The exit program, if enabled at the XFCBOVER exit point, is invoked when a log record is presented to file control for backing out an update to a data set in RLS access mode, after the data set has been used in a batch update despite the existence of retained locks. A consequence of running a batch program while there are retained locks is that a lock that protected a record updated by CICS could have been overridden by a non-RLS batch program.

There is more information about using the XFCBOVER exit, and about the DFH$FCBV sample program, in the comments within the DFH$FCBV source code. The comments also include some suggested extensions that you can make to the sample program to reflect the pattern of batch usage at your installation.

In summary, DFH$FCBV performs the following processing:

- Makes a user trace entry if tracing is active for file control. This has trace point id X'01E0' and traces:
  - An eye-catcher ‘DFH$FCBV ENTRY'
  - The data set name
  - The file control portion of the log record.
- Checks the data set name to see if it is one of those for which it is known that batch programs never update existing records, but only insert new records, or do not make updates at all. The sample program contains a table of such data sets. If this data set is in the table, UERCBCKO is returned. UERCBCKO means that CICS is to perform the backout, despite the override option having been used, because the locked record cannot have been updated by a batch job.

- For all other data sets, it must be assumed that the batch job could have updated the record being backed out. The sample therefore returns UERCNORM, which instructs CICS to take the default action of not backing out the update.
- Exit from the program, making:
  - A user trace entry if tracing is active for file control. This has trace point id X'01E1' and traces:
    - An eye-catcher 'DFH$FCBV EXIT'
    - The data set name
    - Some text: 'Update will be backed out', or 'Update will not be backed out' as appropriate.

Related concepts:

"File control recovery sample exit programs (DFH$FC*)" on page 24
CICS includes three sample file control global user exit programs. The sample programs are DFH$FCBF, DFH$FCBV, DFH$FCLD.

Exit XFCLDEL, file control logical delete exit

Exit XFCLDEL is invoked whenever a WRITE to a VSAM ESDS, or to a BDAM data set, is being backed out. Because these types of data set do not support deletion, you can use XFCLDEL to perform a logical delete by amending the record in some way that flags it as deleted.

Exit-specific parameters

UEPBLOGR
Address of the file control portion of the log record representing the update that is to be backed out by logical deletion. The log record can be mapped using the DSECT DFHFCLGD.

UEPTRANS
Address of the 4-byte transaction id under which the update that is being backed out was made.

UEPTRMNL
Address of the 4-byte terminal id for the terminal or principal facility from which the update that is being backed out was made.

UEPTASK
Address of the 4-byte (packed decimal) task number for the task under which the update that is being backed out was made.

UEPFDATA
Address of a variable-length field containing the data in the file control request. The exit program can amend the record data addressed by this field, marking it in some way that applications can recognize as representing a logically deleted record.

UEPFLEN
Address of a fullword containing the length of the data in the file control request.

Return codes

UERCFAIL
Do not perform the logical delete, and treat this as a backout failure. This is the default action taken if the exit is not enabled.

UERCLDEL
Perform the logical delete by reapplying the updated record.

A return code of UERCPURG is not allowed. There is no need to set a UERCPURG return code, because the conditions under which this exit is invoked should mean that "purged" cannot be returned by any XPI or API calls.

XPI calls

All can be used, but subject to the same caution as for API and SPI calls.

API and SPI calls

Although this exit is allowed to issue API and SPI calls, you should be very careful about which commands you use because the exit is invoked during file backout, which is part of syncpoint phase 2.

It is recommended that you restrict EXEC CICS commands to inquiries, and avoid commands that update CICS resources, because the resources may themselves be in a state of recovery. In particular, the following restrictions apply:

1. Do not issue any recoverable operations.
2. Do not use operations that access systems or resource owners external to this CICS, even if the target resource is non-recoverable.
3. Do not disable or close files, because this could cause further error conditions.
4. It is possible for this exit to be invoked under a different transaction environment from that under which the updates that are being backed out were originally made. If your exit program wants to perform any actions (such as writing a message to the terminal) that require it to be running under the original transaction environment, it must first check the value returned in the RE_ATTACHED_TRANSACTION parameter of a transaction manager INQUIRE_TRANSACTION XPI call.

DFH$FCLD sample global user exit program:

DFH$FCLD provides sample processing for the file control logical delete global user exit, XFCLDEL.

The exit program, if enabled at the XFCLDEL exit point, is invoked when a WRITE to a VSAM ESDS or BDAM data set is being backed out. Because these access methods do not support a physical delete operation, special action must be taken to provide a logical delete function. Normally this involves flagging the record in a way that application programs that use the data set recognize as meaning the record has been deleted.

There is more information about using the XFCLDEL user exit, and about the DFH$FCLD sample program, in the comments within the DFH$FCLD source code.

In summary, DFH$FCLD performs the following processing:

- Makes a user trace entry if tracing is active for file control. This has trace point id X'01F0' and traces:
  - An eye-catcher 'DFH$FCLD ENTRY'
  - The unmarked file control request data
  - The file control portion of the log record.
- Issues an EXEC CICS INQUIRE FILE command to check the access method and type to confirm that the file is a VSAM ESDS or BDAM data set. The logical delete exit should have been invoked only if the file is one of these types.
- For a VSAM ESDS:

  - Flags the record (whose address is passed to the exit in UEPFDATA) as logically deleted. The sample adopts what is probably the most common convention, which is to flag the first byte with a logical delete mark of X'FF'.
  - Takes the normal exit from the program.
- For BDAM:
  - Flags the record (whose address is passed to the exit in UEPFDATA) as logically deleted. The sample adopts a convention for BDAM of flagging the first byte with a logical delete mark of X'C0'.
  - Takes the normal exit from the program.
- For any other combination of access method and type:
  - Does not process the request, and the record is not flagged as deleted
  - Takes the error exit from the program.
- Normal exit from the program:
  - Makes a user trace entry if tracing is active for file control. This has trace point id X'01F1' and traces:
    - An eye-catcher 'DFH$FCLD EXIT OK'
    - An eye-catcher 'RECORD MARKED AS DELETED'
    - The marked file control request data
    - The file control portion of the log record.
  - Returns to CICS with return code UERCLDEL, which instructs CICS to rewrite the marked record and therefore to logically delete it.
- Error exit from the program:
  - Makes a user exception trace entry regardless of the trace setting. This has trace point id X'01F2' and traces:
    - An eye-catcher 'USEREXC'
    - An eye-catcher 'DFH$FCLD EXIT FAIL'
    - The unmarked file control request data
    - The file control portion of the log record.
  - Returns to CICS with return code UERCFAIL, which instructs CICS to regard the logical delete as having failed. (The return code UERCNORM is not intended for use by this exit. Returning UERCNORM has the same effect as UERCFAIL.)

Related concepts:

"File control recovery sample exit programs (DFH$FC*)" on page 24
CICS includes three sample file control global user exit programs. The sample programs are DFH$FCBF, DFH$FCBV, DFH$FCLD.

File control RLS coexistence program exit XFCRLSCO

The XFCRLSCO exit can be called during a request to open a file. Use this exit to allow an application to switch the mode between RLS and read-only non-RLS to access a particular data set.

CICS does not allow an open request to take place for a non-RLS file if an RLS file is already open against the same data set, or if the data set has outstanding RLS work in the system. Also, CICS does not allow an open request to take place for an RLS file if an existing non-RLS file is already open against the base data set. In these situations, if an open request occurs and the non-RLS file is open for read-only access, the XFCRLSCO exit is driven. You can use this exit to decide whether to allow the open request to proceed or to fail in the usual manner with message DFHFC0511 or DFHFC0512.

To switch the access mode, the application can open the data set using a new file with a different access mode. Do not keep the same data set open using both access methods simultaneously over an extended period of time, because CICS does not receive a consistent view of the data set when accessing it concurrently using both RLS and non-RLS files. In particular, CICS cannot get a consistent view if the data set is being updated by the RLS file at the same time as it is being read by the non-RLS file.

If VSAM upgrade set processing occurs while the data set is open using both RLS and non-RLS files, there is an increased risk that read errors might occur because the upgrade processing has not completed on either the base cluster or the associated alternate indexes.

Note:

1. The exit is not driven if the non-RLS file has any updatable SERVREQS set (that is, it allows updates, adds, or deletes).
2. The data set being opened must specify share options SHAREOPTION(2) on the VSAM base cluster. If lower share options are specified, VSAM fails the second open.
3. If static allocation is being used then be sure to specify DISP=SHR, otherwise VSAM fails the open.

When invoked

During a file OPEN request, before the open request is issued to VSAM.

Exit-specific parameters

UEPFILEN

Address of an 8-byte field containing the file name. If the file name is less than 8 characters in length, it is padded with blanks.

UEPDSNAME

Address of a 44-byte field containing the base data set name. If the data set name is less than 44 characters in length, it is padded with blanks.

UEPFSERV

Address of a 1-byte field containing the file SERVREQ indicator. Possible values are:

UEPFRDIM

Read valid indicator.

UEPFUPDIM

Update valid indicator.

UEPFADDIM

Add valid indicator.

UEPFDELIM

Delete valid indicator.

UEPFBRZIM

Browse valid indicator.

UEPFDSACC

Address of a 1-byte field containing the file access method flag. Possible values are:

UEPFVSAM

VSAM file indicator.

UEPFDTBL
Data table file indicator.

UEPFDTUM
User data table file indicator.

UEPFRLS
RLS file indicator.

UEPFCFDT
CFDT file indicator.

UEPRECUR
Address of the halfword recursion level.

Return codes

UERCNORM
Continue processing as normal. The open request fails.

UERCBYP
Allow the open to take place, and bypass the coexistence failure.

XPI calls
You can use XPI commands, but the commands must not result in any state changes to any files.

API and SPI calls
You can use EXEC CICS API and SPI commands, but the commands must not make any state changes to any files. For example, you can use **EXEC CICS INQUIRE FILE**, but not **EXEC CICS SET FILE**, or EXEC CICS API commands against file control which result in state changes to any files.

Good morning message program exit (XGMTEXT)

This exit is invoked before a "good morning" message is sent.

When invoked
Before the good morning message is transmitted.

Exit-specific parameters

UEPTCTTE
Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.

UEPTIOA
Address of the terminal input/output area (TIOA). The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are **not** programming interfaces.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

HTTP client open and send exits: XWBAUTH, XWBOPEN and XWBSNDO

Exits XWBAUTH, XWBOPEN and XWBSNDO are invoked during processing of **EXEC CICS WEB CONVERSE**, **EXEC CICS WEB OPEN**, **EXEC CICS INVOKE SERVICE**, and **EXEC CICS WEB SEND** commands. They are used in making HTTP client requests from CICS as an HTTP client, which is a facility provided by CICS Web support.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

"HTTP client sample exit programs (DFH$WB*)" on page 25
CICS includes four sample programs for use with the Web domain exits XWBOPEN and XWBAUTH. The sample programs are DFH$WBPI, DFH$WBEX, DFH$WBX1, DFH$WBX2.

HTTP client send exit XWBAUTH

With XWBAUTH, you can specify basic authentication credentials (user name and password) for a target server or service provider. XWBAUTH passes them to CICS on request, to create an Authorization header, which is forwarded using HTTP.

When you specify AUTHENTICATE(BASICAUTH) in the **EXEC CICS** WEB SEND (Client) or WEB CONVERSE command, the application can provide a user name and password. If they are not supplied, XWBAUTH is called, providing an alternative way of specifying these credentials.

The user name and password are typically specific to the remote server environment, and might be longer than the standard eight characters used by RACF systems. The user name and password fields can be up to 256 characters in length. The syntax of these fields is not validated.

The host is passed to the user exit program as the UEPHOST parameter, and the path is passed as the UEPPATH parameter. The realm is passed optionally as the UEPREALM parameter. In response, the user exit program returns the user name and password as the UEPUSNM and UEPPSWD parameters.

The following sample exit programs are shipped in the CICS sample library, SDFHSAMP:

- DFH$WBPI
- DFH$WBEX
- DFH$WBX1
- DFH$WBX2
- DFH$WBGA, a copybook to map the global work area used by the DFH$WBPI, DFH$WBX1, DFH$WBX2, and DFH$WBEX samples.

For more information about the client sample exit programs, see *CICS Customization Guide*. For more information about setting up your LDAP profile, see *CICS RACF Security Guide*.

Exit XWBAUTH

When invoked

When the **EXEC CICS** WEB SEND or WEB CONVERSE command specifies AUTHENTICATE(BASICAUTH), but the USERNAME and PASSWORD are not specified.

Exit-specific parameters

UEPHOST (Input supplied by CICS)
The address of a field containing the address of the host name, IPv4, or IPv6 address specified in the HOST option of the WEB OPEN command for the connection. The host name is converted into lowercase characters when it is saved in this field. Your user exit program must take this conversion into account when matching the host name.

UEPHOSTL (Input supplied by CICS)
The address of a field containing the halfword length of the host name.

UEPPATH (Input supplied by CICS)
The address of a field containing the address of the path specified in the PATH option of the WEB SEND or WEB CONVERSE command. The path is mixed case, as it was specified.

UEPPATHL (Input supplied by CICS)
The address of a field containing the halfword length of the path.

UEPREALM (Input supplied by CICS)
The address of a field containing the address of the realm name associated with the target destination, if a realm name was returned in a previous HTTP 401 response from the server.

UEPREALML (Input supplied by CICS)
The address of a field containing the halfword length of the realm name.

UEPAUTHT (Input supplied by CICS)
The address of a 1-byte code that indicates the authentication type. This code is a binary 01, indicating Basic Authentication.

UEPUSNM (Output supplied by user exit)
The address of a fullword field, containing the address of the user name required to access the HTTP server. A predefined address and 64-byte area are created by CICS to store the user name. You can place your user name in this 64-byte area, leaving the address in UEPUSNM unchanged. Alternatively, you can place your user name in your own area and replace the address in UEPUSNM with your user name address. If you create your own user name area, the field can be up to 256 bytes in length.

UEPUSNML (Input supplied by CICS and output supplied by user exit)
The address of a halfword field, which initially contains the length of the buffer address supplied in UEPUSNM. Your user exit program must set the length of this buffer to the user name length, as supplied in UEPUSNM.

UEPPSWD (Output supplied by user exit)
The address of a fullword field, containing the address of the password required to access the HTTP server. A predefined address and 100-byte area are created by CICS to store the password or password phrase. You can place your password in this 100-byte area, leaving the address in UEPPSWD unchanged. Alternatively, you can place your password in your own area and replace the address in UEPPSWD with the address of your password. If you create your own password area, the field can be up to 256 bytes in length.

UEPPSWDL (Input supplied by CICS and output supplied by user exit)
The address of a halfword field, which initially contains the length of the buffer address supplied in UEPPSWD. Your user exit program must set the length of this buffer to the actual password length, as supplied in UEPPSWD.

UEPHOSTT (Input supplied by CICS)
The address of a 1-byte code that indicates the host type contained in the UEPHOST parameter.

Binary 01 indicates host name, binary 02 indicates an IPv4 address, and binary 03 indicates an IPv6 address.

Return codes

UERCNORM
The exit has successfully returned a user name and password.

UERCBYP
The exit cannot identify a user name and password. An Authorization header is not sent.

UERCERR
The exit cannot identify a user name and password. The WEB SEND (Client) or WEB CONVERSE command must be stopped.

XPI calls
All XPI calls can be used.

API and SPI commands
All API and SPI commands can be used, except for **EXEC CICS SHUTDOWN** and **EXEC CICS XCTL**.

Typical use of the LDAP XPI functions by XWBAUTH:

The expected use of the DFHDDAPX functions (in association with the XWBAUTH global user exit) include opening and closing an LDAP session, browsing results for credentials, scanning and locating results, closing the browse, returning the correct value and closing the search.

BIND_LDAP
Establishes a session with an LDAP server. Used once on the first call to the global user exit XWBAUTH. The LDAP session token is stored in XWBAUTH's global work area (if one is provided) for use by subsequent calls to LDAP_SEARCH.

UNBIND_LDAP
Releases the connection with the LDAP server. This function is only required during CICS shutdown processing. This function can be used during the XSTERM (system termination) global user exit.

SEARCH_LDAP
Searches for credentials, specifying an LDAP distinguished name, that identifies the URL and realm of the required user information. Distinguished name is specified in the following format:

```
racfcid=uuuuuuuu, ibm-httprealm=rrrrrrrr, labeledURI=xxxxxxxx, cn=BasicAuth
```

where:

- `uuuuuuuu` is the current userid, obtained from the XWBAUTH parameter, UEPUSER.
- `rrrrrrrr` is the HTTP 401 realm, obtained from the XWBAUTH parameter, UEPREALM (if this exists).
- `xxxxxxxx` is the target URL, obtained by concatenating `http://` with the hostname from the XWBAUTH parameter, UEPHOST, and the path from the XWBAUTH parameter, UEPPATH.
- `cn=BasicAuth` is an arbitrary suffix that is configured into the LDAP server for storing Basic Authentication credentials.

START_BROWSE_RESULTS
Starts scanning the results returned by SEARCH_LDAP.

GET_NEXT_ENTRY
Locates the next result entry in a series of entries returned by SEARCH_LDAP. Typically, the URL specified in SEARCH_LDAP will locate a unique entry and the GET_NEXT_ENTRY function is not used.

GET_NEXT_ATTRIBUTE
Locates the next attribute in the current result entry. Typically, specific attributes will be selected and the GET_NEXT_ATTRIBUTE function is not used.

END_BROWSE_RESULTS
Ends the browse session started by SEARCH_LDAP.

GET_ATTRIBUTE_VALUE
Returns the values for various attributes of the target distinguished name. For XWBAUTH, these attributes values are the username and password, stored in the attributes `uid` and `userpassword`. XWBAUTH returns these attribute values as credentials.

FREE_SEARCH_RESULTS
Closes the search initiated by SEARCH_LDAP and releases associated storage.

HTTP client open exit XWBOPEN

With XWBOPEN, you can specify proxy servers that are used for HTTP requests by CICS as an HTTP client. You can also apply a security policy to the host name specified for those requests.

XWBOPEN is called during processing of an **EXEC CICS WEB OPEN** command, which is used by an application program to open a connection with a server. XWBOPEN is also called during processing of an **EXEC CICS INVOKE SERVICE** command.

CICS does not have any requirements concerning the use (or otherwise) of proxy servers for HTTP requests by CICS as an HTTP client, and CICS does not apply any security policy for those requests. You have to set up these facilities if they are required by your system or organization.

The **EXEC CICS** WEB OPEN command instructs the CICS web domain to open a connection with a server. XWBOPEN is called before the connection is opened. The host name for the connection (for example, `www.example.com`), which is specified by the HOST option on the **EXEC CICS** WEB OPEN command, is passed as the UEPHOST parameter to the user exit program for checking. At this point, you can use the user exit program for two purposes:

- To determine whether the HTTP request needs to use a proxy server, and to return the name of any proxy server that is required. If a proxy server is needed, return code UERCPROX is used, and the name of the proxy server is returned to the CICS web domain, in the buffer identified by UEPPROXY, and used to make the connection to the server. If no proxy server is needed, return code UERCNORM is used.
- To apply a security policy to the host name. Return code UERCBARR indicates that access to the host is not permitted and a NOTAUTH response is returned to the WEB OPEN command. The application programmer must stop trying to open that connection. If you want to apply a security policy for individual resources, as well as (or instead of) for the host, use the XWBSNDO user exit on the **EXEC CICS** WEB SEND and **EXEC CICS** WEB CONVERSE commands to apply a security policy to the path component of the URL.

The XWBOPEN user exit does not support the use of **EXEC CICS** commands.

The sample programs DFH$WBPI and DFH$WBEX, with the associated copybook DFH$WBGA, show you how to set up proxy server information or a security policy in a global work area. For example, if all the requests from your CICS system must use a single proxy server, you can specify the proxy server name as an initialization parameter. If you use a number of proxy servers or want to apply a security policy to different host names, you can load or build a table that matches host names to appropriate proxy servers or marks them as barred, which can then be used as a lookup table during processing of the **EXEC CICS** WEB OPEN command. The sample programs can be run during program list table post initialization (PLTPI) processing or at any point before you expect the **EXEC CICS** WEB OPEN command to be used.

Exit XWBOPEN

When invoked

During processing of an **EXEC CICS WEB OPEN** or **EXEC CICS INVOKE SERVICE** command.

Exit-specific parameters

UEPHOST (Input supplied by CICS)
The address of a field containing the host name, IPv4, or IPv6 address specified in the HOST option of the WEB OPEN command.

Note: The host name is converted into lowercase when it is saved in this field. Your user exit program must take into account this conversion when matching the host name.

UEPHOSTL (Input supplied by CICS)
The address of a field containing the halfword length of the host name.

UEPPROXY (Output supplied by user exit)
The address of a field containing the address that points to the proxy server name. The proxy server name must be in URL format. On input to the user exit program, the parameter is set to the address of a field containing the address of a 2046-byte area. You can place the proxy server name in this area and leave the address in UEPPROXY unchanged. Alternatively, you can place the proxy server name in your own area and replace the address in UEPPROXY with the address of a field containing the address of your own area.

UEPPROXYL (Output supplied by user exit)
The address of a field containing the halfword length of the proxy server name.

UEPHOSTT (Input supplied by CICS)
The address of a 1-byte code that indicates the host type contained in the UEPHOST parameter.

Note: Binary 01 indicates host name, binary 02 indicates an IPv4 address, and binary 03 indicates an IPv6 address.

Return codes

UERCNORM
A proxy server is not needed for this HTTP request, and the host name is not barred.

UERCPROX
A proxy server is needed for this HTTP request. UEPPROXY has been set to the name of the required proxy server, and UEPPROXYL has been set to the length of the proxy server name.

UERCBARR
The host name of the server is barred.

UERCERR
An error occurred in exit processing.

XPI calls

All XPI calls can be used.

API and SPI commands

No **EXEC CICS** commands can be used.

HTTP client send exit XWBSNDO

With XWBSNDO, you can specify a security policy for HTTP requests by CICS as an HTTP client. XWBSNDO is called during processing of an **EXEC CICS** WEB SEND or **EXEC CICS** WEB CONVERSE command. The host name and path information are passed to the exit, and a security policy can be applied to either or both of these components.

CICS does not apply any security policy for HTTP requests by CICS as an HTTP client; you must set up this facility if it is required by your system or organization.

You can use the XWBOPEN exit on the WEB OPEN command to bar access to a whole host. You use the XWBSNDO exit to do the same or to bar access to specific paths in a host. To bar access to a whole host, using the XWBOPEN exit saves time, because the application program cannot open the connection and so does not waste time creating the request that must be sent. The host name is provided to the XWBSNDO exit so that you can differentiate between identical paths used by different hosts.

If chunked transfer-coding is being used for the HTTP request, XWBSNDO is called only on the first WEB SEND command for the chunked message.

The XWBSNDO user exit does not support the use of **EXEC CICS** commands.

The host is passed to the user exit program as the UEPHOST parameter, and the path is passed as the UEPPATH parameter. Return code UERCNORM indicates that the path is permitted, and return code UERCBARR indicates that the path is not permitted. If the path is not permitted, a NOTAUTH response is returned to the WEB SEND or WEB CONVERSE command, and the application programmer handles this response by closing the connection with a WEB CLOSE command.

Exit XWBSNDO

When invoked

During processing of an **EXEC CICS** WEB SEND or **EXEC CICS** WEB CONVERSE command for an HTTP request by CICS as an HTTP client. A client request is indicated by the use of the SESSTOKEN parameter on the WEB SEND command.

Exit-specific parameters

UEPHOST
The address of a field containing the host name, IPv4, or IPv6 address specified in the HOST option of the WEB OPEN command for the connection.

Note: The host name is converted into lowercase when it is saved in this field. Your user exit program must take this conversion into account when matching the host name.

UEPHOSTL
The address of a field containing the halfword length of the host name.

UEPPATH
The address of a field containing the path specified in the PATH option of the WEB SEND command. The path is in mixed case, as it was specified.

UEPPATHL
The address of a field containing the halfword length of the path.

UEPHOSTT
The address of a 1-byte code that indicates the host type contained in the UEPHOST parameter.

Note: Binary 01 indicates host name, binary 02 indicates an IPv4 address, and binary 03 indicates an IPv6 address.

Return codes

UERCNORM
The path is permitted.

UERCBARR
The path is not permitted.

XPI calls
All XPI calls can be used.

API and SPI commands
No **EXEC CICS** commands can be used.

Intersystem communication program exits, XISCONA, XISLCLQ, and XISQLCL

The three exits in the intersystem communication program allow you to control the length of intersystem queues.

You can use several methods to control the length of intersystem queues. For a description of the available methods, see the *CICS Intercommunication Guide*.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

The XISCONA exit

The purpose of XISCONA is to help you prevent the performance problems that can occur when function shipping or DPL requests awaiting free sessions for a non-IPIC connection are queued in the issuing region. The exit permits you to control the number of outstanding ALLOCATE requests by allowing you to reject any function shipping or DPL request that would otherwise be queued.

Important: Use the XZIQUE exit in the z/OS Communications Server working-set module to control the length of intersystem queues, rather than XISCONA. XZIQUE provides more functions, and is of more general use than XISCONA (it is driven for function shipping, DPL, transaction routing, and distributed transaction processing requests, whereas XISCONA is driven only for function shipping and DPL). If you enable both exits, XZIQUE and XISCONA could both be driven for function shipping and DPL requests, which is not recommended.

If you already have an XISCONA exit program, you may be able to modify it for use at the XZIQUE exit point.

"Contention winner" is the terminology used for LU6.2 connections. The XISCONA exit applies also to MRO and LU6.1 connections: in these, the SEND sessions (defined in the session definitions) are used first for ALLOCATE requests; when all SEND sessions are in use, queuing starts.

Function shipping and DPL requests for a resource-owning region are queued by default if all bound contention winner sessions are busy, so that no sessions are immediately available. If the resource-owning region is unresponsive (for example, if it is a file-owning region, it may be waiting for a system journal to be archived), the queue can become so long that the performance of the issuing region is severely impaired. Further, if the issuing region is an application-owning region, its impaired performance can spread back to the terminal-owning region.

To control the queuing of function shipping and DPL requests, use the XISCONA exit to tell CICS, whenever a session cannot be allocated immediately, whether to queue the request, or to return 'SYSIDERR' to the application. The exit works like this:

1. If the XISCONA exit program is **not** active, CICS queues the request when necessary.
2. If the exit program is active, it is invoked *only if all bound contention winner sessions are in use*. For other failures (for example, 'Mode name not found' or 'Out of service'), CICS bypasses the exit and returns to the application.
3. If it is invoked, your exit program must decide whether or not to queue the request by analyzing the statistics provided through the user exit parameter list. Your exit program could:
   - Stipulate that queuing is never to be used. This is the simplest way to code the exit, and avoids complexities of tuning. It should be effective if you define enough contention winner sessions to handle the peak transaction load for the connection. If you suppress all queuing, you must specify AUTOCONNECT(YES) on the SESSIONS definition, because the queuing mechanism no longer binds sessions for you.

     With this approach, a danger arises if you base your estimate of required sessions on average conditions and the transaction load subsequently varies widely; when CICS cannot use queuing to cope with the variation, users may suffer transaction abends when there is no significant problem in the resource-owning region.
   - Examine the number of requests currently in the queue. The program could, for example, stop queuing when the number exceeds 120% of the maximum number of sessions. You could use this approach to cope with intermittent stoppages in the resource-owning region.

     You could use a table of thresholds for the connections in your system, with values determined from previous experience of queuing problems. Alternatively, you could use the EXEC CICS interface in a separate program to inquire about the state of the connection, and pass the information in a work area to the XISCONA exit program.
   - Examine the type of request and the resource being accessed (which can be discovered by examining the request parameter list). The program could, for example, reject file read requests but queue file updates.

   Note: Because a failure of the exit program could affect system availability, it is recommended that you make the logic of your program as simple as possible, thus reducing the possibility of errors.

There are some problems that XISCONA cannot solve. For example, if you have specified both a large number of sessions and a large value for MXT, CICS may develop the short-on-storage (SOS) condition *before* XISCONA is invoked because there are no further sessions available.

The sample XISCONA global user exit program, DFHXIS:

A CICS-supplied sample exit program, DFHXIS,shows one way of limiting the queue of ALLOCATE requests, based on the information passed to the program.

For more information about the sample global user exit programs, see "Sample global user exit programs" on page 19.

Related concepts:

"Function-shipping/DPL queue control sample exit program (DFHXIS)" on page 25

CICS includes a sample global user exit program that controls the queueing of function-shipping and DPL requests. The sample program is DFHXIS.

Exit:

Exit XISCONA is invoked when a function shipping or DPL request is about to be queued because all bound contention winner sessions to the remote region are in use.

When invoked

When a function shipping or DPL request is about to be queued because all bound contention winner sessions to the remote region are in use.

Note: For DPL requests that are routed dynamically, the dynamic routing program is invoked before XISCONA. If there are no free sessions the routing program may choose not to queue a DPL request; in these circumstances, XISCONA is not invoked. For information about the dynamic routing of DPL requests, see Dynamically routing DPL requests in Getting started.

Exit-specific parameters

UEPISPCA

Address of a parameter list containing the following fields. You can map the parameter list using the DSECT DFHXISDS.

UEPCONST

Address of the Connection statistics record.

Connection statistics records are of type STICONSR (STID value 52). Your exit program can map the record using the DSECT DFHA14DS. See notes.

UEPMODST

Address of the Mode Entry statistics record, or zero. A Mode Entry statistics record is built *only* if:

- The connection-type is LU6.2 (see field UEPCONTY).
- The profile DFHCICSF (which is always used for function shipping) defines a specific MODENAME to be used in the allocation of LU6.2 sessions.

Mode Entry statistics records are of type STICONMR (STID value 76). Your exit program can map the record (if present) using the DSECT DFHA20DS.

UEPEIPPL

Address of the request parameter list.

UEPCONTY
A 1-byte field indicating the connection-type. Possible values are:

UEPMRO (X'80')
Request for an MRO connection

UEPLU6 (X'40')
Request for an LU6.1 connection

UEPLUC (X'20')
Request for an LU6.2 connection.

UEPNETNM
An 8-character field containing the NETNAME for the connection; that is, the identifier (applid) of the remote CICS region or system.

Note:

1. The general format of statistics records is described in Chapter 26, "CICS statistics record format," on page 653.
2. For a list of statistics record-types and their associated copy books, see Figure 81 on page 657.
3. For a description of the fields in Connection and Mode Entry statistics records, see CICS statistics in DSECTs and DFHSTUP report, in the *CICS Performance Guide*.

Return codes

UERCAQUE
Queue the request. This is the default.

UERCAPUR
Do not queue the request, unless local queuing is possible.

XPI calls
All can be used.

Important

There is no 'UERCNORM' return code at this exit point, because the exit is invoked after a failure. The choice is whether or not to take the system default action of queuing the request.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The XISLCLQ exit

XISLCLQ is used for EXEC CICS START NOCHECK requests that are scheduled for a non-IPIC connection.

XISLCLQ is invoked, if enabled, under any of the following circumstances:

- The remote system is not in service.
- A connection to the remote system cannot be established.
- No sessions are immediately available, and your XISCONA exit program has specified that the request is not to be queued in the issuing region.

Note that this exit is invoked *only* if the request to be shipped is of type EXEC CICS START NOCHECK. For EXEC CICS requests other than those with the NOCHECK option (which is only available on START commands) the SYSIDERR condition is raised in the application program.

You can use the exit to specify whether the failed request is to be locally queued, to be run when the connection is reestablished.

Local queues are recovered when you perform a system restart.

Exit XISLCLQ:

Exit XISLCLQ is invoked after a function shipping request of type EXEC CICS START NOCHECK has failed because the remote system is not in service, a connection to the remote system cannot be established, or no sessions are immediately available, and your XISCONA exit program has specified that the request is not to be queued in the issuing region.

When invoked

After a function shipping request of type EXEC CICS START NOCHECK has failed because the remote system is not in service, a connection to the remote system cannot be established, or no sessions are immediately available, and your XISCONA exit program has specified that the request is not to be queued in the issuing region.

Exit-specific parameters

UEPISPP

Address of a parameter list that contains:

UEPTCTSE

Address of the relevant terminal control table system entry. The TCT system entry can be mapped using the DSECT DFHTCTTE.

UEPXXTE

Address of the local transaction name, or 0 if SYSID was specified in the command.

Note: Your program can use the transaction manager XPI call INQUIRE_TRANDEF to obtain details of the local transaction (see “The INQUIRE_TRANDEF call” on page 858).

UEPPLIST

Address of the parameter list for the command.

Note: No DSECT is provided for this parameter list. You have to code your own DSECT to access the named fields.

Return codes

UERCSYS
Take the system action. This is determined by the value of the LOCALQ attribute in the local TRANSACTION definition for the remote transaction:

LOCALQ(YES)
The request is queued locally.

LOCALQ(NO)
'SYSIDERR' is returned to the application program.

UERCQUE
Queue the request locally (overriding the LOCALQ(NO) attribute, if specified).

UERCIGN
Override the LOCALQ(YES) attribute, if specified, and return with 'SYSIDERR'.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Important

There is no 'UERCNORM' return code at this exit point, because the exit is invoked after a failure. The choice is whether to take the system default action or to handle the error in some other way.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The XISQLCL exit

You can use the XISQLCL exit for EXEC CICS START NOCHECK commands that are scheduled for an IPIC connection.

It is invoked, if enabled, under any of the following circumstances:

- The IPIC connection is not acquired.
- A session is not available and CICS does not queue the request for a new session.

XISQLCL allows you to decide whether to add the request to a local queue or to return with an error response.

Local queues are recovered when you perform a system restart.

The sample XISQLCL exit program, DFH£XISL:

You use the XISQLCL sample global user exit program DFH£XISL to control the queueing of START NOCHECK requests that are scheduled for an IPIC connection.

For more information about the sample global user exit programs, see "Sample global user exit programs" on page 19.

Exit XISQLCL:

Exit XISQLCL is invoked after a function shipping request of a **START NOCHECK** or **START NOCHECK PROTECT** command over IPIC fails because the remote system is not in service, a connection to the remote system cannot be established, or no sessions are immediately available, and your XISQUE exit program specifies that the request is not queued in the issuing region.

Exit-specific parameters

The DSECT, DFHXILDS, is provided for this parameter list.

UEPISQPL

Address of a parameter list that contains the following fields:

UEPPLIST

The address of the parameter list for the command.

UEPQLEN

A halfword binary field containing the number of items currently on the queue.

UEPIPCNM

The eight-byte name of the IPCONN.

UEPTRID

The four-byte identifier of the local transaction name, or blanks if SYSID is specified in the command. Your program can use the transaction manager XPI call, INQUIRE_TRANDEF, to obtain details of the local transaction. .

Return codes

UERCSYS

Take the system action. This action is determined by the value of the LOCALQ attribute in the local TRANSACTION definition for the remote transaction:

LOCALQ(YES)

The request is queued locally.

LOCALQ(NO)

A SYSIDERR error message is returned to the application program.

UERCQUE

Queue the request locally, overriding the LOCALQ(NO) attribute, if specified.

UERCIGN

Override the LOCALQ(YES) attribute, if specified, and return with a SYSIDERR response.

UERCPURG

Task purged during XPI call.

XPI calls
All can be used.

Important

There is no UERCNORM return code at this exit point, because the exit is invoked after a failure. You must choose whether to take the system default action or to handle the error in some other way.

The sample XISQLCL global user exit program, DFH$XISL, is provided.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Interval control program exits XICREQ, XICEXP, and XICTENF

You can use some XPI calls in exit programs invoked from the interval control program. However, when any of these exits are invoked for expiry analysis, any actions that delay the execution of the interval control program can have adverse effects on other transactions that are waiting for intervals to expire.

You can determine whether the exits have been invoked for expiry analysis by examining the type-of-request field, TCAICTR, a copy of which is pointed to by the UEPICRQ1 exit-specific parameter.

The XICREQ exit is invoked by internal requests made by CICS code, as well as by requests made by applications. DFHXRSP issues an interval control WAIT every 2 seconds; this means that any interval control exit programs are also invoked every 2 seconds.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XICREQ

This exit is invoked at the beginning of the interval control program, before request analysis.

When invoked
At the beginning of the interval control program, before request analysis.

Exit-specific parameters

UEPICQID
Address of an 8-byte field containing the request ID parameter on request. See notes 1 and 2.

UEPICTID
Address of a 4-byte field containing the terminal ID, if any, specified on an EXEC CICS START command. See notes 1 and 2.

UEPICTI
Address of 4 bytes containing the transaction ID specified on an EXEC CICS START command. See notes 1 and 2.

UEPICRQ1
Address of a 1-byte field containing a copy of TCAICTR, the first request code field for requests to the interval control program.

UEPICRQ2
Address of a 1-byte field containing a copy of TCAICTR2, the second request code field for requests to the interval control program.

UEPICRT
Address of a 4-byte field containing the expiry time or interval, in packed decimal format. The value is in the form 0HHMMSSF, where H=hours, M=minutes, S=seconds, and F is a positive sign.

Note:

1. The contents of the fields addressed by UEPICQID and UEPICTID are unpredictable if the associated data items were not specified on the request. You must test the copy of TCAICTR to determine whether they contain meaningful values.
2. Your exit program can change the values of the fields addressed by UEPICQID, UEPICTID UEPICTI, and UEPICRT. Changing the values of the fields addressed by UEPICRQ1 or UEPICRQ2 has no effect.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls

The following must not be used:

- ADD_SUSPEND
- DELETE_SUSPEND
- DEQUEUE
- ENQUEUE
- RESUME
- SUSPEND
- WAIT_MVS.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell

you whether the call was successful.

Exit XICEXP

This exit is invoked after an interval control time interval has expired.

When invoked

After an interval control time interval has expired.

Exit-specific parameters

UEPICE

Address of the interval control element (ICE) that has just expired. The ICE can be mapped using the DSECT DFHICEDS.

Return codes

UERCNORM

Continue processing.

UERCPURG

Task purged during XPI call.

XPI calls

The following must not be used:

- ADD_SUSPEND
- DELETE_SUSPEND
- DEQUEUE
- ENQUEUE
- RESUME
- SUSPEND
- WAIT_MVS.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XICTENF

This exit is invoked from the interval control program.

When invoked

This exit relates to the 'terminal not known' condition. For more information see "'Terminal not known' condition exits XALTENF and XICTENF" on page 290.

Interval control EXEC interface program exits (XICEREQ, XICERES, and XICEREQC)

These exits are invoked when interactions with interval control programs occur.

XICEREQ

XICEREQ is invoked on entry to the interval control program before CICS processes an interval control request. Using XICEREQ, you can:

- Analyze the request to determine its type, the keywords specified, and their values.
- Modify any value specified by the request before the command is executed.
- Set return codes to specify that either:
  - CICS should continue with the request, modified or unmodified.
  - CICS should bypass the request. (Note that if you set this return code, you must also set up return codes for the EXEC interface block (EIB), as if you had processed the request yourself.)

Note: The XICEREQ exit is invoked by internal requests made by CICS code, as well as by requests made by applications.

XICERES

XICERES is invoked by the interval control program, before CICS processes a non-terminal-related EXEC CICS START request that has been dynamically routed to this region.

Note that XICERES is invoked:

- After exit XICEREQ and before XICEREQC (if these exits are enabled). This means that:
  - If an XICEREQ exit program chooses to bypass the request, XICERES is not invoked, even if it is enabled.
  - If an XICEREQ exit program modifies the request, XICERES must deal with the modified request.
- On the *target* region—that is, the region to which the START request has been routed.
- Only if the routing region—the region on which the routing program runs—supports the "resource unavailable" condition (RESUNAVAIL). To support the "resource unavailable" condition, the routing region must be a supported release of CICS TS.
- Only if it is enabled. It is strongly recommended that you enable this exit only in application-owning regions to which non-terminal-related **EXEC CICS START** requests may be dynamically routed.
- By internal requests made by CICS code, as well as by requests made by applications.

The XICERES exit is *not* invoked:

- For statically-routed requests.
- For terminal-related EXEC CICS START requests. (These always execute in the terminal-owning region and cannot be routed.)
- For dynamically-routed *transactions* - only dynamically-routed (non-terminal-related) START requests cause the exit to be invoked. Thus, a dynamically-routed transaction that was initiated by a terminal-related EXEC CICS START command does not cause the exit to be invoked.
- If it is disabled.
- If an XICEREQ exit program chooses to bypass the request.

You can use XICERES to check that all resources required by the transaction to be started are available on the target region. If, for example, the transaction is disabled, or a required file is missing, your exit program can give the distributed routing program the opportunity to route the request to a different region. To do this, set a return code of UERCRESU. This causes CICS to:

1. Set the DYRERROR field of the distributed routing program's communications area to 'F'—resource unavailable.

2. Reinvoke the routing program, on the routing region, for route selection failure.
3. Return a RESUNAVAIL condition on the EXEC CICS START command executed by the mirror on the target region. (This condition is not returned to the application program.)

CICS ignores any changes made by the exit program to the values of any of the exit parameters. Your exit program can set a return code, but not change any parameters.

For guidance information about dynamically routing non-terminal-related EXEC CICS START requests, see the *CICS Intercommunication Guide*. For information about writing a distributed routing program to route non-terminal-related **EXEC CICS START REQUESTS**, see "Routing non-terminal-related START requests" on page 614.

XICEREQC

XICEREQC is invoked after an interval control program request has completed. Using XICEREQC, you can:

- Analyze the request, to determine its type, the keywords specified, and their values.
- Set return codes for the EIB.

Note: The XICEREQC exit is invoked by internal requests made by CICS code, as well as by requests made by applications.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Parameters passed to each of the exits

CICS passes ten types of address parameter to the exits.

- Address of the command-level parameter structure (UEPCLPS)
- Address of a token (UEPICTOK) used to pass 4 bytes of data from XICEREQ to XICEREQC
- Addresses of copies of six return code segments, resource, date, and time information from the EIB
- Address of a token (UEPTSTOK) that is valid throughout the life of a task
- Address of an exit recursion count (UEPRECUR).

Exit XICEREQ

Exit XICEREQ is invoked before CICS processes an interval control API request.

Exit-specific parameters

UEPCLPS
Address of the command-level parameter structure. See "The UEPCLPS exit-specific parameter" on page 189.

UEPICTOK
Address of a 4-byte token to be passed to XICEREQC. This allows you, for example, to pass a work area to exit XICEREQC.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code

'EIBRCODE'. For details of EIB return codes, refer to EIB fields, in the *CICS Application Programming Reference* manual.

UEPRESP
Address of a 4-byte binary copy of the EIB response code 'EIBRESP'.

UEPRESP2
Address of a 4-byte binary copy of the EIB response code 'EIBRESP2'.

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

UEPDATE
Address of a fullword copy of the EIB date value, EIBDATE.

UEPTIME
Address of a fullword copy of the EIB time value, EIBTIME.

Return codes

UERCNORM
Continue processing.

UERCBYP
The interval control EXEC interface program should ignore this request.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used. You can also use EXEC CICS API commands at this user exit.

Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, you are recommended to use the EXEC CICS GETMAIN and FREEMAIN commands instead.

API and SPI commands
All can be used, except for:
EXEC CICS SHUTDOWN
EXEC CICS XCTL

Note: Take care when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when an interval control request is issued from the XICEREQ exit. Use of the recursion counter UEPRECUR is recommended.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XICERES

Exit XICERES is invoked by the interval control program, before processing of a non-terminal-related EXEC CICS START request that has been dynamically routed to this region, where the routing region supports the "resource unavailable" condition (RESUNAVAIL).

Exit-specific parameters

Note: CICS ignores any changes made by the exit program to the values of any of the exit parameters. Your exit program can set a return code, but not change any parameters.

UEPCLPS
Address of the command-level parameter structure. See "The UEPCLPS exit-specific parameter" on page 189.

UEPICTOK
Address of a 4-byte token to be passed to XICEREQC.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code 'EIBRCODE'. For details of EIB return codes, refer to EIB fields, in the *CICS Application Programming Reference* manual.

UEPRESP
Address of a 4-byte binary copy of the EIB response code 'EIBRESP'.

UEPRESP2
Address of a 4-byte binary copy of the EIB response code 'EIBRESP2'.

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEPRECUR
Address of a halfword recursion counter. Because the XICERES exit can never be called recursively in the same transaction, the value of this field is always 0.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

UEPDATE
Address of a fullword copy of the EIB date value, EIBDATE.

UEPTIME
Address of a fullword copy of the EIB time value, EIBTIME.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

UERCRESU
A required resource is unavailable. Setting this value causes CICS to reject the routed request, and to return a value of 'F' (resource unavailable) in the DYRERROR field of the routing program's communications area.

XPI calls
All can be used. You can also use EXEC CICS API commands at this user exit.

Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, you are recommended to use the EXEC CICS GETMAIN and FREEMAIN commands instead.

API and SPI commands
All except EXEC CICS SHUTDOWN and EXEC CICS XCTL can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XICEREQC

Exit XICEREQC is invoked after an interval control API request has completed, and before return from the interval control EXEC interface program.

When invoked
After an interval control API request has completed, and before return from the interval control EXEC interface program.

Exit-specific parameters

UEPCLPS
Address of the command-level parameter structure. See "The UEPCLPS exit-specific parameter" on page 189.

UEPICTOK
Address of a 4-byte token passed from XICEREQ. This allows XICEREQ to, for example, pass a work area to XICEREQC.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code 'EIBRCODE'. For details of EIB return codes, refer to the *CICS Application Programming Reference* .

UEPRESP
Address of a 4-byte binary copy of the EIB response code 'EIBRESP'.

UEPRESP2
Address of a 4-byte binary copy of the EIB response code 'EIBRESP2'.

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked and increments for each recursive call.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

UEPDATE
Address of a fullword copy of the EIB date value, EIBDATE.

UEPTIME
Address of a fullword copy of the EIB time value, EIBTIME.

UEP_IC_REMOTE_SYSTEM
If the request is to be sent to a remote region, is the address of an area containing the 4-byte name of the remote region. (The remote region may have been specified by, for example, the SYSID option of the START command, workload management, or the REMOTESYSTEM option of the TRANSACTION definition.)

If the request is to be executed on the local region, this parameter is the address of a 4-byte area containing blanks.

UEP_IC_REMOTE_NAME
If the transaction is to be executed in a remote system, is the address of an area containing the name of the transaction, as it is known in the remote system.

The remote system may be another CICS region, or an IMS system. If UEP_IC_REMOTE_SYSTEM names a CICS region, the name is 1 through 4 characters long. If UEP_IC_REMOTE_SYSTEM names an IMS system, the name is 1 through 8 characters long. IMS uses 8-character names: if UEP_IC_REMOTE_NAME has fewer than 8 characters, IMS translates it into a usable format.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, you are recommended to use the EXEC CICS GETMAIN and FREEMAIN commands instead.

API and SPI commands
All can be used, except for:

```
EXEC CICS SHUTDOWN
```

```
EXEC CICS XCTL
```

Note: Take care when issuing recursive commands. For example, you must avoid entering a loop when issuing an interval control request from the XICEREQC exit. Use of the recursion counter UEPRECUR is recommended.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The command-level parameter structure

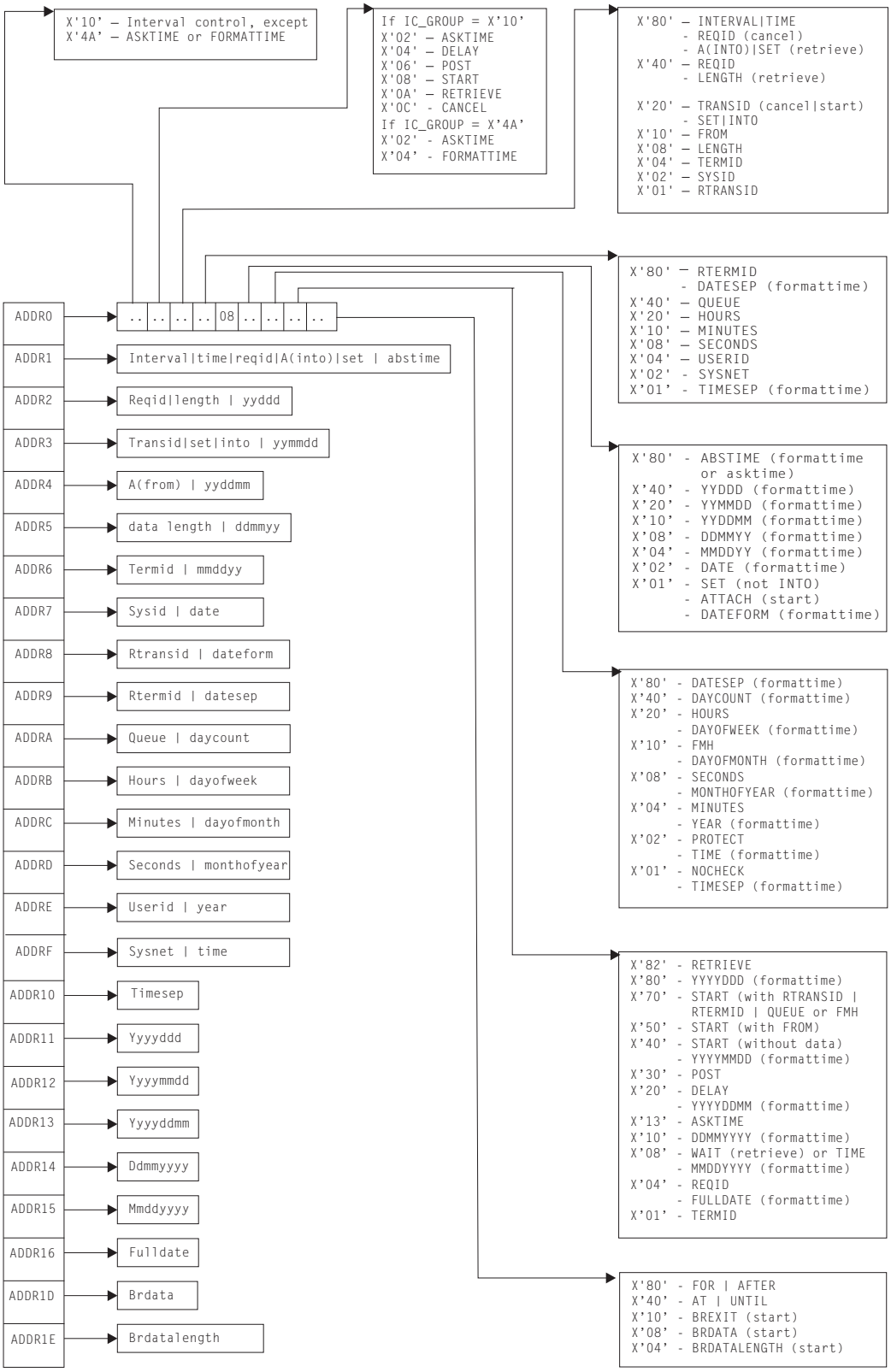


Figure 7. The command-level parameter structure for interval control

The command-level parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of a 9-byte area that describes the type of request and identifies each keyword specified with

the request. The remaining addresses point to pieces of data associated with the request. For example, the second address points to the interval for START requests.

You can examine the EID to determine the type of request and the keywords specified. You can examine the other parameters in the list to determine the values of the keywords. You can also modify values of keywords specified on the request. For example, you could change the SYSID specified in the request.

End of parameter list indicator

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

For example, if the parameter list specifies only the first four addresses (IC_ADDR0, the address of the EID, to IC_ADDR3, the address of the name of the transaction named in a START request), the high-order bit is set on in IPC_ADDR3. If you extend the parameter list by setting the address of a SYSID in IC_ADDR7, you must unset the high-order bit in IC_ADDR3 and set it on in IC_ADDR7 instead.

The maximum size of parameter list is supplied to the exit, thus allowing your exit program to add any parameters not already specified without needing to first obtain more storage.

The original parameter list, as it was before XICEREQ was invoked, is restored after the completion of XICEREQC. It follows that the execution diagnostic facility (EDF) displays the original command before **and** after execution: **EDF does not display any changes made by the exit**.

The UEPCLPS exit-specific parameter:

The UEPCLPS exit-specific parameter is included in both exit XICEREQ and exit XICEREQC. It is the address of the command-level parameter structure.

The command-level parameter structure contains 26 addresses, IC_ADDR0 through IC_ADDR1F. It is defined in the DSECT IC_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHICUED.

The command-level parameter list is made up as follows:

IC_ADDR0
is the address of a 9-byte area called the EXEC interface descriptor (EID), which is made up as follows:

- **IC_GROUP**
- **IC_FUNCT**
- **IC_BITS1**
- **IC_BITS2**
- **IC_BITS3**
- **IC_EIDOPT5**
- **IC_EIDOPT6**
- **IC_EIDOPT7**

- **IC_EIDOPT8**

IC_GROUP

X'10' This is an interval control request.

X'4A' This is an ASKTIME or FORMATTIME command.

IC_FUNCT

One byte that defines the type of request.

If IC_GROUP = X'10':

X'02' ASKTIME

X'04' DELAY

X'06' POST

X'08' START

X'0A' RETRIEVE

X'0C' CANCEL

If IC_GROUP = X'4A':

X'02' ASKTIME

X'04' FORMATTIME

IC_BITS1

Existence bits that define which arguments were specified. To obtain the argument associated with a keyword, you need to use the appropriate address from the command-level parameter structure. Before using this address, you must check the associated existence bit. If the existence bit is set off, the argument was not specified in the request and the address should not be used.

X'80' Set if the request contains INTERVAL or TIME arguments, or if a CANCEL request specifies REQID, or if a RETRIEVE request specifies SET or INTO. If set, **IC_ADDR1** is meaningful.

X'40' Set if the request other than CANCEL specifies REQID or if a RETRIEVE request specifies LENGTH. If set, **IC_ADDR2** is meaningful.

X'20' Set if the request specifies TRANSID or if a request other than RETRIEVE specifies SET or INTO. If set, **IC_ADDR3** is meaningful.

X'10' Set if the request specifies FROM. If set, **IC_ADDR4** is meaningful.

X'08' Set if a request other than RETRIEVE specifies LENGTH. If set, **IC_ADDR5** is meaningful.

X'04' Set if the request specifies TERMID. If set, **IC_ADDR6** is meaningful.

X'02' Set if the request specifies SYSID. If set, **IC_ADDR7** is meaningful.

X'01' Set if the request specifies RTRANSID. If set, **IC_ADDR8** is meaningful.

IC_BITS2

Further argument existence bits.

X'80' Set if the request specifies RTERMID, or if a FORMATTIME request specifies DATESEP. If set, **IC_ADDR9** is meaningful.

X'40' Set if the request specifies QUEUE. If set, **IC_ADDRA** is meaningful.

X'20' Set if the request specifies HOURS. If set, **IC_ADDRB** is meaningful.

X'10' Set if the request specifies MINUTES. If set, **IC_ADDRC** is meaningful.

X'08' Set if the request specifies SECONDS. If set, **IC_ADDRD** is meaningful.

X'04' Set if the request specifies USERID. If set, **IC_ADDRE** is meaningful.

X'02' Set if the request specifies SYSNET. If set, **IC_ADDRF** is meaningful.

X'01' Set if a FORMATTIME request specifies TIMESEP. If set, **IC_ADDR10** is meaningful.

IC_BITS3
One byte not used by interval control.

IC_EIDOPT5
Indicates whether certain keywords were specified on the request.

X'80' ABSTIME was specified on a FORMATTIME or ASKTIME command.

X'40' YYDDD was specified on a FORMATTIME command.

X'20' YYMMDD was specified on a FORMATTIME command.

X'10' YYDDMM was specified on a FORMATTIME command.

X'08' DDMMYY was specified on a FORMATTIME command.

X'04' MMDDYY was specified on a FORMATTIME command.

X'02' DATE was specified on a FORMATTIME command.

X'01' On a RETRIEVE command, SET (and not INTO) was specified. On a START command, ATTACH was specified. On a FORMATTIME command, DATEFORM was specified. You cannot modify this field in your user exit.

IC_EIDOPT6
Existence bits that indicate whether certain keywords were specified on the request.

X'80' DATESEP was specified on a FORMATTIME command.

X'40' DAYCOUNT was specified on a FORMATTIME command.

X'20' DAYOFWEEK was specified on a FORMATTIME command, or HOURS was specified.

X'10' DAYOFMONTH was specified on a FORMATTIME command, or FMH was specified.

X'08' MONTHOFYEAR was specified on a FORMATTIME command, or SECONDS was specified.

X'04' YEAR was specified on a FORMATTIME command, or MINUTES was specified.

X'02' TIME was specified on a FORMATTIME command, or PROTECT was specified.

X'01' TIMESEP was specified on a FORMATTIME command, or NOCHECK was specified.

IC_EIDOPT7

Indicates whether certain functions or keywords were specified on the request.

X'F0' CANCEL specified.

X'82' RETRIEVE specified.

X'80' YYYYDD specified on a FORMATTIME command.

X'40' YYYYMMDD specified on a FORMATTIME command, or START specified.

X'30' POST specified.

X'20' YYYYDDMM specified on a FORMATTIME command, or DELAY, RTRANSID, RTERMID, or QUEUE specified, and/or FMH.

X'13' ASKTIME specified.

X'10' DDMMYYYY specified on a FORMATTIME command, or FROM, RTRANSID, or RTERMID specified, and/or QUEUE.

X'08' MMDDYYYY specified on a FORMATTIME command, or TIME or WAIT specified.

X'04' FULLDATE specified on a FORMATTIME command, or REQID specified.

X'01' TERMID specified.

IC_EIDOPT8

Indicates whether certain keywords were specified on the request.

X'80' FOR or AFTER specified.

X'40' AT or UNTIL specified.

X'10' BREXIT specified.

X'08' BRDATA specified.

X'04' BRDATALENGTH specified.

X'02' CHANNEL specified on a START command.

IC_ADDR1

is the address of one of the following:

- An 8-byte area containing the value of the INTERVAL keyword (or TIME keyword if **IC_EIDOPT7** indicates that TIME is specified).
- An 8-byte area containing the value of REQID (if the request is CANCEL).
- An 8-byte area containing the value of the ABSTIME keyword.
- Data returned for INTO (if the request is RETRIEVE, and if **IC_EIDOPT5** indicates that this is not SET).
- A 4-byte address returned for SET (if the request is RETRIEVE and **IC_EIDOPT5** indicates that this is SET).

IC_ADDR2
is the address of one of the following:

- An 8-byte area containing the value of REQID (if the request is DELAY, POST or START).
- A halfword containing the value of LENGTH (if the request is RETRIEVE).

  Warning: For requests that specify INTO, do not change the value of LENGTH to a value greater than that specified by the application. To do so causes a storage overlay in the application.
- An area containing the value of YYDD.

IC_ADDR3
is the address of one of the following:

- An area containing the value of TRANSID (if the request is CANCEL or START).
- A 4-byte address returned for SET (if the request is START or POST and **IC_EIDOPT5** indicates that this is SET).
- An area containing the value of YYMMDD.

IC_ADDR4
is the address of one of the following:

- An area containing the data from FROM.
- An area containing the value of YYDDMM.

IC_ADDR5
is the address of one of the following:

- An area containing the halfword value of LENGTH.

  Warning: For requests that specify INTO, do not change the value of LENGTH to a value greater than that specified by the application. To do so causes a storage overlay in the application.
- An area containing the value of DDMMYY.

IC_ADDR6
is the address of one of the following:

- An area containing the value of TERMID.
- An area containing the value of MMDDYY.

IC_ADDR7
is the address of one of the following:

- An area containing the value of SYSID.
- An area containing the value of DATE.

IC_ADDR8
is the address of one of the following:

- An area containing the value of RTRANSID.
- An area containing the value of DATEFORM.

IC_ADDR9
is the address of one of the following:

- An area containing the value of RTERMID.
- An area containing the value of DATESEP.

IC_ADDRA
is the address of one of the following:

- An area containing the value of QUEUE.

- A fullword containing the value of DAYCOUNT.

IC_ADDRB
is the address of one of the following:
- An area containing the value of HOURS.
- A fullword containing the value of DAYOFWEEK.

IC_ADDRC
is the address of one of the following:
- An area containing the value of MINUTES.
- A fullword containing the value of DAYOFMONTH.

IC_ADDRD
is the address of one of the following:
- An area containing the value of SECONDS.
- A fullword containing the value of MONTHOFYEAR.

IC_ADDRE
is the address of one of the following:
- An area containing the value of USERID.
- A fullword containing the value of YEAR.

IC_ADDRF
is the address of one of the following:
- An 8–byte area containing the value of SYSNET.
- An area containing the value of TIME.

IC_ADDR10
is the address of a 1–byte area containing the value of TIMESEP.

IC_ADDR11
is the address of an area containing the value of YYYYDDD.

IC_ADDR12
is the address of an area containing the value of YYYYMMDD.

IC_ADDR13
is the address of an area containing the value of YYYYDDMM.

IC_ADDR14
is the address of an area containing the value of DDMMYYYY.

IC_ADDR15
is the address of an area containing the value of MMDDYYYY.

IC_ADDR16
is the address of an area containing the value of FULLDATE.

IC_ADDR1D
is the address of an area containing the value of BRDATA.

IC_ADDR1E
is the address of a fullword containing the value of BRDATALENGTH.

IC_ADDR1F
is the address of a 16-byte area containing the value of CHANNEL.

Modifying fields in the command-level parameter structure:

Some fields that are passed to interval control are used as input to the request, some are used as output fields, and some are used for both input and output. The method your user exit program uses to modify a field depends on the usage of the field.

The following are always input fields:

- INTERVAL
- TIME
- REQID
- FROM
- TERMID
- SYSID
- HOURS
- MINUTES
- SECONDS
- USERID
- CHANNEL

The following are always output fields:

- DATE
- DATEFORM
- DAYCOUNT
- DAYOFMONTH
- DAYOFWEEK
- DDMMYY
- DDMMYYYY
- FULLDATE
- INTO
- MMDDYY
- MMDDYYYY
- MONTHOFYEAR
- SET
- TIME
- YEAR
- YYDDD
- YYDDMM
- YYMMDD
- YYYYDDD
- YYYYDDMM
- YYYYMMDD

The following are input fields on a START request and output fields on a RETRIEVE request:

- RTRANSID
- RTERMID

- QUEUE

LENGTH is an input field on a START request, an output field on a RETRIEVE with SET specified, and an input/output field on a RETRIEVE with INTO specified.

ABSTIME is an input field on a FORMATTIME request, and an output field on an ASKTIME request. DATESEP and TIMESEP can be input fields on a FORMATTIME request.

Modifying input fields:

The correct method of modifying an input field is to create a new copy of it, and to change the address in the command-level parameter list to point to your new data.

Note: You must never modify an input field by altering the data that is pointed to by the command-level parameter list. To do so would corrupt storage belonging to the application program and would cause a failure when the program attempted to reuse the field.

Modifying output fields:

You modify an output field by altering the data to which the command-level parameter list points.

The technique described in "Modifying input fields" is not suitable for modifying output fields because the results would be returned to the new area instead of the application's area, and would be invisible to the application.

In the case of an output field, you can modify the application's data in place because the application is expecting the field to be modified anyway.

Modifying the EID:

It is not possible to modify the EID to make major changes to requests, such as changing a DELAY request to a START request. However, you can make minor changes to requests, such as turning on the existence bit for SYSID so that the request can be changed into one that is shipped to a remote system.

Some interval control commands use 2 bits in the EID to indicate a single keyword; the EXEC CICS START command, for example, uses 2 bits to indicate TERMID. The first bit, in IC_BITS1, indicates that ADDR6 in the command parameter list is valid (ADDR6 points to TERMID) and the second, in IC_EIDOPT7, is the keyword existence bit to show that the TERMID keyword was specified on the command.

Where this occurs you must ensure that both bit settings are changed (consistently) if you want to modify these commands from within a user exit program, or the results will be unpredictable.

The list that follows shows the bits in the EID that **can** be modified. Any attempt to modify any other part of the EID is ignored.

IC_BITS1

X'80' The existence bit for REQID (if the request is CANCEL)

X'40' The existence bit for LENGTH (if the request is RETRIEVE) or REQID

X'10' The existence bit for FROM

X'08' The existence bit for LENGTH

X'04' The existence bit for TERMID

X'02' The existence bit for SYSID

X'01' The existence bit for RTRANSID.

IC_BITS2

X'80' The existence bit for RTERMID

X'40' The existence bit for QUEUE

X'20' The existence bit for HOURS

X'10' The existence bit for MINUTES

X'08' The existence bit for SECONDS.

IC_EIDOPT6

X'20' The secondary existence bit for HOURS

X'10' The existence bit for FMH

X'08' The secondary existence bit for SECONDS

X'04' The secondary existence bit for MINUTES

X'02' The existence bit for PROTECT

X'01' The existence bit for NOCHECK.

IC_EIDOPT7

Bits in IC_EIDOPT7 should only be modified within the same functional group; that is, only those existence bits defined as valid for a START request should be set on a START request.

ASKTIME requests

X'13' ASKTIME request. This value is fixed for all ASKTIME requests, and should not be modified.

DELAY requests

X'20' DELAY request

X'08' TIME specified

X'04' REQID specified.

POST requests

X'30' POST request

X'08' TIME specified

X'04' REQID specified.

START requests

X'40' START request (without DATA)

X'50' START with DATA request

X'70' START with one or more of RTRANSID, RTERMID, QUEUE, or FMH specified.

X'08' TIME specified

X'04' REQID specified

X'01' TERMID specified.

RETRIEVE requests

X'82' RETRIEVE request.

CANCEL requests

X'F0' CANCEL request

X'04' REQID specified.

IC_EIDOPT8

X'20' Unused by CICS.

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the interval control request only.

Note: Your user exit program is prevented from making major changes to the EID. However, you must take great care when making the minor modifications that **are** permitted.

Using the interval control request token UEPICTOK:

UEPICTOK provides the address of a 4-byte area that you can use to pass information between the XICEREQ and XICEREQC user exits for the same interval control request.

For example, the address of a piece of storage that is obtained by the XICEREQ user exit, which is to be freed by the XICEREQC exit, can be passed in the UEPICTOK field.

UEPICTOK is usable only for the duration of a single interval control request, because its contents might be destroyed at the end of the request. If you need to pass information between successive invocations of a global user exit, you can use task token UEPTSTOK to do so. For more information about UEPTSTOK, see “Using the task token UEPTSTOK” on page 19.

EXEC interface block (EIB):

Copies of EIBRSRCE, EIBRCODE, EIBRESP, and EIBRESP2 are passed to the exit so you can modify/set completion and resource information in XICEREQ and XICEREQC, or examine completion and resource information in XICEREQC.

You can update the copies of EIBRSRCE, EIBRCODE, EIBRESP, and EIBRESP2 that you are given in the parameter list. Interval control copies your values into the real EIB after the completion of XICEREQC; or if you specify a return code of ‘bypass' in XICEREQ.

You must set valid interval control responses. You must set all three of EIBRCODE, EIBRESP, and EIBRESP2 to a consistent set of values, such as would be set by CICS interval control to describe a valid completion. **CICS does not police the consistency of EIBRCODE, EIBRESP, and EIBRESP2**. However, if EIBRCODE is set to a non-zero value and EIBRESP is set to zero, CICS overrides EIBRESP with a

non-zero value. To aid you in setting the values of EIBRCODE, EIBRESP, and EIBRESP2, the values used by interval control are specified in DFHICUED.

Example of how XICEREQ and XICEREQC can be used:

In this example, XICEREQ and XICEREQC are used to route START requests to a number of different CICS regions to provide a simple load balancing mechanism. The example shows only the capabilities of the exits; it is not intended to indicate an ideal way of achieving the function.

In XICEREQ:

1. Scan the global work area (GWA) to locate a suitable CICS region (for example, the region currently processing the least number of START requests).
2. Having decided which system to route the request to, increment the use count for this system.
3. Obtain a 4-byte area in which to store the SYSID for this request. This can be allocated from the GWA to avoid issuing a GETMAIN. If the area is obtained by issuing a GETMAIN, set UEPICTOK to the address of the storage obtained.
4. Set IC_ADDR7 to be the address of the 4-byte area so that XICEREQC can also use this area.
5. If setting IC_ADDR7 now makes it the last address, set the high-order bit in the address, and reset the high-order bit in what was previously the last address.
6. Set the X'02' existence bit on in IC_BITS1 to indicate that a SYSID is specified.
7. Return to CICS.

In XICEREQC:

1. Scan the global work area (GWA) and locate the entry for the CICS region specified in the SYSID parameter.
2. Decrement the use count for this system.
3. If a GETMAIN was issued in XICEREQ to obtain an area to hold the SYSID, issue a FREEMAIN for the address held in UEPICTOK.
4. Return to CICS.

Example and sample programs

CICS supplies two programs for use at the XICEREQ exit:

- DFH$XTSE, supplied as a softcopy listing only (not as a source code file), is an example program that shows how to modify fields in the command-level parameter structure passed to all the EXEC interface exits. DFH$XTSE is listed in Appendix G, "The example program for the XTSEREQ global user exit, DFH$XTSE," on page 917.
- DFH$ICCN is a sample program for use in a distributed routing environment, where you want to cancel a previously-issued interval control request but have no way of knowing to which region to direct the CANCEL. For examples of situations which DFH$ICCN is designed to cope with, see Canceling interval control requests, in the *CICS Intercommunication Guide*.

Related concepts:

"Interval control EXEC interface sample exit program (DFH$ICCN)" on page 28
CICS includes an interval control EXEC interface sample exit program. The sample

program is DFH$ICCN.

Loader domain exits XLDLOAD and XLDELETE

There are two global user exits in the loader domain. XLDLOAD is invoked when a new instance of a program is loaded into storage, before the program is made available for use.

XLDELETE is invoked after an instance of a program is released by CICS and before the program is freed from storage.

For LPA-resident programs, the exits are still invoked when a program is acquired or released, even though the program is not physically loaded or freed.

These are both information-only exits. Any changes made to the exit parameters by the exit program are ignored by CICS, as is any return code which it sets.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XLDLOAD

This exit is invoked for a program instance brought into storage, before the program becomes available.

When invoked

After an instance of a program is brought into storage, and before the program is made available for use.

Exit-specific parameters

UEPPROGN
Address of an 8-character field containing the name of the program that is being loaded.

UEPPROGL
Address of a 4-byte field containing the length, in bytes, of the program that is being loaded.

UEPLDPT
Address of a 4-byte field containing the address at which the program has been loaded.

UEPENTRY
Address of a 4-byte field containing the address of the program's entry point.

UEPTRANID
Zero, or the address of a 4-byte field containing the transaction ID which applied when the exit was invoked.

UEPUSER
Zero, or the address of an 8-byte field containing the userid in control at the time the exit was invoked.

UEPTERM
Zero, or the address of a 4-byte field containing the terminal name associated with the transaction under which the exit was invoked.

UEPPROG
Zero, or the address of an 8-character field containing the name of the program that was in control at the time the exit was invoked.

UEPLDCTXT
Zero, or the address of a 140-byte field containing the application context when a private program belonging to an application is loaded. The field contains:

1. The private resource name, padded with spaces to 8 characters.
2. The platform name, padded with spaces to 64 characters.
3. The application name, padded with spaces to 64 characters.
4. The major version number for the application, which is a fullword binary value.
5. The minor version number for the application, which is a fullword binary value.
6. The micro version number for the application, which is a fullword binary value.

CICS supplies a DSECT named **DFHUEACD** which maps this information. For more information about **DFHUEACD**, see UEACD - User exit application context in Data Areas.

Return codes

UERCNORM
Continue processing.

XPI calls
Must not be used.

API and SPI calls
Must not be used.

Exit XLDELETE

This exit is invoked when a program instance is released by CICS, before the program is freed from storage.

When invoked
After an instance of a program is released by CICS, and before the program is freed from storage.

Exit-specific parameters

UEPPROGN
Address of an 8-character field containing the name of the program that is being freed.

UEPPROGL
Address of a 4-byte field containing the length, in bytes, of the program that is being freed.

UEPLDPT
Address of a 4-byte field containing the address at which the program resides in storage.

UEPENTRY
Address of a 4-byte field containing the address of the program's entry point.

UEPTRANID
Zero, or the address of a 4-byte field containing the transaction ID which applied when the exit was invoked.

UEPUSER
Zero, or the address of an 8-byte field containing the userid in control at the time the exit was invoked.

UEPTERM
Zero, or the address of a 4-byte field containing the terminal name associated with the transaction under which the exit was invoked.

UEPPROG
Zero, or the address of an 8-character field containing the name of the program that was in control at the time the exit was invoked.

UEPLDCTXT
Zero, or the address of a 140-byte field containing the application context when a private program belonging to an application is deleted. The field contains:

1. The private resource name, padded with spaces to 8 characters.
2. The platform name, padded with spaces to 64 characters.
3. The application name, padded with spaces to 64 characters.
4. The major version number for the application, which is a fullword binary value.
5. The minor version number for the application, which is a fullword binary value.
6. The micro version number for the application, which is a fullword binary value.

CICS supplies a DSECT named **DFHUEACD** which maps this information. For more information about **DFHUEACD**, see UEACD - User exit application context in Data Areas.

Return codes

UERCNORM
Continue processing.

XPI calls
Must not be used.

API and SPI calls
Must not be used.

Log manager domain exit XLGSTRM

There is one exit point, XLGSTRM, in the log manager domain. You can use XLGSTRM to modify a request to MVS to create a new log stream. You can change the model log stream name and other parameters before they are passed to the MVS system logger.

If a log stream connection request from CICS to the MVS system logger fails because the log stream is not defined to MVS, CICS issues a request to the MVS system logger to create the log stream dynamically, using a model log stream definition.

The model log stream name that CICS passes to MVS depends on whether the journal name refers to the system log or a CICS general log, as follows:

CICS system logs

&sysname.LSN_last_qualifier.MODEL

&sysname is the MVS symbol that resolves to the system name of the MVS image. *LSN_last_qualifier* is the last qualifier of the log stream name as specified on the JOURNALMODEL resource definition.

If you do not provide a JOURNALMODEL resource definition for DFHLOG and DFHSHUNT, or if you use the CICS definitions supplied in group DFHLGMOD, the model log stream names default to *&sysname*.DFHLOG.MODEL and *&sysname*.DFHSHUNT.MODEL.

For example, if a CICS region issues a request to create a log stream for its primary system log, and CICS is running in an MVS image with a sysid of MV10 and using the default JOURNALMODEL definition, the MVS system logger expects to find a model log stream named MV10.DFHLOG.MODEL.

If the system name of the MVS image starts with a numeric character and is less than 8 characters long, CICS prefixes it with a "C", so that the model log stream name becomes *C&sysname.LSN_last_qualifier*. This is because the MVS system logger rejects log stream names that begin with a numeric. If the system name of the MVS image starts with a numeric but is already 8 characters long (the maximum), CICS does not add the "C" prefix, which means that the MVS system logger will reject the default model log stream name. However, your global user exit program can change the model log stream name.

CICS general logs

LSN_qualifier_1.LSN_qualifier2.MODEL. The defaults for these two qualifiers are the CICS region userid and the CICS region APPLID, but they can be user-defined values specified in a JOURNALMODEL resource definition.

For example, if the CICS region userid is CICSHT## and the APPLID is CICSHTA1, the default model name is CICSHT##.CICSHTA1.MODEL.

The following information is passed to an XLGSTRM global user exit program:

- The name of the log stream to be defined
- The default model log stream name
- A system log flag
- The MVS system logger IXGINVNT parameter list.

Your exit program can amend the model stream name by updating the field pointed to by the UEPMLSN exit-specific parameter. Here is an example of how your exit program can change the model stream name:

```
L     R3,UEPMLSN          R3 = address of stream name
MVC   0(26,R3),=CL26'NEW.MODEL.NAME'
```

By updating the field pointed to by the UEPIXG parameter, your exit program can amend the IXGINVNT macro parameter list used by the MVS system logger to define the log stream. Use the IXGINVNT MF=M form of the macro, which allows the exit to specify the log stream attributes to be used. Here is an example of how your exit program can change the structure name:

```
L     R9,UEPIXG
IXGINVNT REQUEST=DEFINE,
      TYPE=LOGSTREAM,
```

```
                STRUCTNAME=NEW_STRUCTURE,
                MF=(M,(R9),NOCHECK)
    ...
NEW_STRUCTURE DC CL16'LOG_SYSTEST_009'
```

You do not need to code the list and execute forms of the IXGINVNT macro, or include the IXGCON or IXGANSAA macros in your exit—these are provided by the CICS code which issues the DEFINE request.

For information about the IXGINVNT service, see the *z/OS MVS Programming: Assembler Services Guide*.

An XLGSTRM global user exit program can set explicit attributes for the log stream definition, and can also set a return code that causes the log stream definition to be bypassed.

Note: If you want XLGSTRM to intercept the connection of the CICS system logs, you must enable your exit program in a first-phase PLT program.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XLGSTRM

Exit XLGSTRM is invoked after the CICS log manager detects that a log stream does not exist and before it calls the MVS system logger to define the log stream dynamically.

Exit-specific parameters

UEPTRANID
The address of the 4-byte transaction id.

UEPUSER
The address of the 8-byte userid associated with the transaction if the current task is a user task.

UEPTERM
The address of the 4-byte terminal id associated with the transaction, if any.

UEPPROG
The address of the 8-byte application program name for this transaction, if any.

UEPLSN
Address of a 26-character field containing the name of the log stream to be defined.

Your exit program should not modify the name of the logstream. On return from the exit, CICS ignores any changes to the contents of the field addressed by UEPLSN. JOURNALMODEL definitions are provided to cater for log stream name selection.

UEPMLSN
Address of a 26-character field specifying the name of the model log stream to be used to provide the attributes for the new log

stream. This field is modifiable to allow the global user exit program to specify a different model log stream name from the one generated by CICS.

UEPIXG
Address of the IXGINVNT macro parameter list for use by the MVS system logger to define the log stream. Using the MF=M form of the IXGINVNT macro, the global user exit program can specify the log stream attributes to be used.

For details of the IXGINVNT macro, see the *z/OS MVS Programming: Authorized Assembler Services Guide*.

UEPLGTYP
Address of a 1-byte field indicating whether the log stream being created is for a system log or a general log. Valid values are:

UEPSYSLG
The log stream is for a CICS system log.

UEPGENLG
The log stream is for a general log (a forward recovery log, a user journal, or auto-journal).

Return codes

UERCNORM
CICS continues and attempts to define the log stream.

UERCBYP
CICS does not attempt to define the log stream. The process that was attempting to use the log stream may fail (for example, a data set open).

XPI calls
All can be used.

API and SPI commands
Must not be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Example of how to use the XLGSTRM exit

The XLGSTRM exit is used for selecting alternative model log streams.

Suppose that 200 CICS regions are running on 20 MVS images; To avoid having to define explicitly each log stream used by each CICS region, you decide to use model definitions. Log streams will be defined to MVS dynamically on their first usage, with an XLGSTRM exit program being used to select from alternative model log streams. This is how it might work:

1. On an initial start of a CICS region, the INITPARM system initialization parameter specifies:

```
INITPARM=(Exit_enabler_pgmname=nnn)
```

where:

- `Exit_enabler_pgmname` is the name of the program that enables the XLGSTRM user exit program.
- `nnn` is a number that identifies a group of CICS regions that share the same set of log stream models.

2. The program that enables the XLGSTRM user exit program issues an EXEC CICS ASSIGN INITPARM command to retrieve the value `nnn`, and places it in the exit program's global work area.
3. When the region tries to connect to its system log, because the log stream is not defined the XLGSTRM exit program is invoked. The exit program selects model CICS.DFHLOG.MODELnnn.

The sample program for the XLGSTRM exit, DFH$LGLS

The sample program, DFH$LGLS, shows you how to access and change some of the parameters passed to an XLGSTRM exit program.

Specifically, the program:

- Changes the model log stream name pointed to by the UEPMLSN exit-specific parameter
- Uses the IXGINVNT macro to change the value of the HIGHOFFLOAD parameter in the log stream definition parameter list pointed to by the UEPIXG exit-specific parameter.

Note: To run the sample program “as is”, you must first create a model log stream called 'CICSAD01.DEPT0001.MODEL100'. However, you will probably want to tailor the sample to suit your own environment. The code contains comments to help you do this.

Related concepts:

“Log manager domain sample exit program (DFH$LGLS)” on page 28
CICS includes a sample global user exit program for the XLGSTRM exit point. The sample program is DFH$LGLS.

Message domain exit XMEOUT

You can use the XMEOUT exit to suppress or reroute CICS and CICSPlex SM messages that use the CICS message domain.

Your exit program has the following restrictions:

- It can suppress or reroute only messages sent to the system console or to transient data queues. It cannot suppress or reroute messages sent to terminal operators. XMEOUT is not invoked for messages sent to terminal operators.
- It can suppress or reroute only messages that use the message domain. You can determine which CICS messages this applies to from CICS messages in Reference -> Diagnostics. The description of each message that can invoke XMEOUT contains a list of message inserts; if no message inserts are listed for a message, the message cannot invoke the exit. For example, message DFHDX8320 can invoke XMEOUT, but message DFHDU0205 does not.

  For CICSPlex SM, XMEOUT is invoked only for messages that have a destination of EYULOG, because these are the messages that use the message domain. You can determine which messages this applies to from CICSPlex SM messages in Reference -> Diagnostics.

Note: CICSPlex SM messages that invoke the XMEOUT exit can be rerouted or suppressed only from the joblog or console, not from the EYULOG.

- It cannot reroute or suppress CICSPlex SM Web User interface messages.
- It cannot change the text of a message, or change the message inserts. If it attempts to do so, CICS ignores the changes.
- It cannot suppress or reroute messages issued during the early stages of CICS initialization (because the exit cannot be enabled then).
- It cannot reroute a message to transient data (TD) queues during CICS shutdown, unless the original message destination included one or more transient data queues. If it attempts to do so, the message is routed to its original destination, and message DFHME0120I is issued to the console. The user exit program cannot reroute message DFHME0120I, but it can suppress this message.

  This restriction is necessary because the message domain must handle messages during CICS shutdown even after the transient data queue function has ended.

  To discover whether CICS shutdown has started, your exit program can check for the first instance of message DFHME0120. It can stop rerouting messages to TD queues after DFHME0120 has been issued.

Note: If a message is being rerouted to a transient data queue and the transient data request fails, the message is lost. The MEME exception trace point ID X'0328' is written. The interpretation string of this trace entry explains why the transient data request failed.

Important

Because of the danger of recursion, your XMEOUT exit program must not try to reroute the following messages:

- Any DFHTDxxxx messages, which are produced by the transient data program.
- User domain messages in the range DFHUS0002 toDFHUS0006, plus message DFHUS0150.
- Transaction manager messages DFHXM0212, DFHXM0213, DFHXM0304, and DFHXM0308.
- Application messages DFHAP0001, DFHAP0002, DFHAP0004, DFHAP0601, DFHAP0602, and DFHAP0603.
- Any user domain (DFHUSxxxx) messages to an intrapartition queue defined with a TRIGLEV value of anything other than zero, if the messages are produced while the user domain is performing error recovery processing.

The message definition template contains an indicator called *noreroute*. This indicator is set on if the message that is being issued cannot be rerouted to a transient data queue by the XMEOUT exit program. The address of the indicator is passed to XMEOUT in the UEPNRTE exit-specific parameter. Your exit program can check the value of the indicator before deciding whether to reroute a particular message.

Note: If the exit program tries to reroute an ineligible message, the message domain inhibits the rerouting and issues the message to the console instead, along with message DFHME0137.

Each message that is affected by this restriction is identified by a note in CICS messages in Reference -> Diagnostics.

It is possible to pass APPLID (the application identifier) as an optional parameter in a message. However, the APPLID that is inserted in a message might not be the APPLID of the current CICS system. For example, when a CICSPlex SM MAS message is routed to a CMAS, the APPLID of the MAS system might be passed, so the message contains the APPLID of the MAS system and not the current system (the CMAS).

Your exit program can suppress or reroute messages by altering the values held in the addresses pointed to by the following fields of the parameter list. Your program cannot change any other sets of values.

- UEPMROU (route codes)
- UEPMNRC (number of route codes)
- UEPMTDQ (transient data queue names)
- UEPMNTD (number of TDQs)

Using XMEOUT to monitor DFHAP1900 message

Message DFHAP1900 is issued when a change is made to the CICS system configuration by certain system programming interface commands. These commands are **SET**, **PERFORM**, **ENABLE**, **DISABLE**, or **RESYNC**. The commands are written to the transient data queue CADS. The DFHAP1900 messages can provide auditing of dynamic configuration changes and also aid problem determination. For more information, see SPI commands that can be audited in Developing system programs.

Note: Do not issue SPI commands in the exit which result in a DFHAP1900 message, because a recursion in the exit can occur.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XMEOUT

This exit is invoked before a message domain sends a CICS message.

When invoked

Before the message domain sends a CICS message to its destination.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Address of the 4-byte ID of the terminal under which the current transaction is running. If the current transaction is not associated with a terminal, the addressed field contains hexadecimal zeroes.

UEPPROG
Address of the 8-byte application program name, or nulls if there is no current application.

UEPMNUM
Address of a 4-byte field containing the message number.For CICSPlex® SM messages, this field contains binary zeros.

UEPMDOM
Address of a 2-byte field containing the domain identifier of the CICS message.For CICSPlex SM messages, this field contains binary zeros.

UEPMROU
Address of an array of up to 28 route codes. Route codes must be numbers in the range 1 through 28.

UEPMNRC
Address of a halfword containing the number of route codes in the route code array.

UEPMTDQ
Address of an array of up to 25 transient data queue names to which the message is to be sent. TD queue names must consist of 4 alphanumeric characters.

UEPMNTD
Address of a halfword containing the number of TDQs in the queues array.

UEPINSN
Address of a 2-byte field containing the number of message inserts.

UEPINSA
Address of an array, each element of which contains information about a single message insert. The size of the array depends on the number of inserts. Each array element has the following structure:

```
INSERT_FORMAT_P  DS  A  Address of the 1-byte insert
                        type-code, which has one of
                        the following hexadecimal values:
                        0    Not present
                        1    Character
                        2    Hexadecimal
                        3    Decimal
                        4    The insert is a number
                             representing one item in
                             a list of options.
                             (See the example below.)
 INSERT_P         DS  A  Address of the message insert
 INSERT_LENGTH_P  DS  A  Address of a fullword contain-
                         ing the length of the insert
 INSERT_TYPE_P    DS  A  Reserved.
```

You can find the order of the inserts in the array from the entry for the particular message in the *CICS Messages and Codes Vol 1* manual. For example,

DFHFC0531 *date time applid* Automatic journal *journal journalname*, opened for file *filename* is not of type MVS. Module *module*.

The XMEOUT inserts are *date, time, applid*, *journal*, *journalname*, *filename*, and *module*. The fourth insert (*journal*) is the number specified for JOURNAL on the file definition.

UEPNRTE
Address of 1-character flag indicating whether or not the message can be rerouted by XMEOUT. The possible values are:

C'0:' The message can be routed.

C'1:' The message cannot be routed.

UEPCPID
Address of a 3-byte product ID. The possible values are:

DFH CICS messages.

EYU CICSPlex SM messages.

UEPCPDOM
Address of a 2-byte field containing the domain identifier of the message.

UEPCPNUM
Address of a 4-byte field containing the message number.

UEPCPSEV
Address of the message severity code.

Return codes

UERCNORM
Continue processing.

UERCBYP
Suppress the message for all destinations.

Note that CICSPlex SM messages cannot be suppressed. For these messages, a response of UERCBYP is treated as UERCNORM.

XPI calls
WAIT_MVS can be used. **Do not use any other calls**.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The sample XMEOUT global user exit programs

CICS includes six sample programs that show how to use the XMEOUT exit to suppress or reroute messages.

DFH$SXP1
Suppress a message by message number

DFH$SXP2
Suppress a message by destination route code

DFH$SXP3
Suppress a message destined for the CSCS transient data queue (which receives signon and sign-off messages)

DFH$SXP4
Reroute a console message to a TDQ

DFH$SXP5
Reroute a TDQ message to another TDQ

DFH$SXP6
Reroute a TDQ message to a console.

Related concepts:

"Message domain sample exit programs (DFH$SXP*)" on page 29
CICS includes six sample programs that are called at the XMEOUT exit to complete specified tasks such as rerouting a console message to a transient data queue. The sample programs are DFH$SXP1, DFH$SXP2, DFH$SXP3, DFH$SXP4, DFH$SXP5, DFH$SXP6.

Monitoring domain exit (XMNOUT)

This exit is invoked before monitoring records are written to SMF or to the record buffers.

XMNOUT is invoked at the following event points:

- Before an exception class monitoring record is passed to SMF
- Before a performance class monitoring record is written to the performance record buffer
- Before a transaction resource monitoring record is written to the transaction resource record buffer

Note: If performance class and transaction resource monitoring are both active in your CICS region, XMNOUT can be invoked twice for the same event. For example, if the event is end-of-task and CICS has both performance class data and transaction resource data to move to the appropriate buffer, XMNOUT is invoked once for each monitoring record type.
You can use this exit to examine the record, to suppress its output to SMF, or to change the data it contains. You must ensure that any changes you make do not conflict with the dictionary description of the data.

You can also add data to performance class data records. To do this you must define dummy user event-monitoring points (EMPs) in the monitoring control table (MCT) to reserve data fields of the required size and type.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XMNOUT

This exit is invoked before monitoring records are written to SMF or buffered for subsequent writing to SMF.

When invoked
XMNOUT is invoked in these circumstances:

- Before an exception class monitoring record is written to SMF
- Before a performance class monitoring record is buffered for a later write to SMF
- Before a transaction resource monitoring record is buffered for a later write to SMF

- Before an identity class monitoring record is buffered for a later write to SMF

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. This field is not available at task termination.

UEPUSER
Address of the 8-byte user ID. This field is not available at task termination.

UEPTERM
Address of the 4-byte terminal ID. This field is not available at task termination.

UEPPROG
Address of the 8-byte application program name. This field is not available at task termination.

UEPDICT
Address of the dictionary. The sequence of dictionary entries is mapped by the DSECT generated from the macro DFHMCTDR. This field has meaning only for performance class records. If the monitoring record type is exception class (type 4), transaction resource (type 5), or identity class (type 6), this field is set to 0. See parameter UEPMRTYP.

UEPDICTE
Address of the fullword number of dictionary entries. This field has meaning only for performance class records. If the monitoring record type is exception class (type 4), transaction resource (type 5), or identity class (type 6), this field is set to 0. See parameter UEPMRTYP.

UEPFCL
Address of the field connector list, containing a series of halfword connector values. This field has meaning only for performance class records. If the monitoring record type is exception class (type 4), transaction resource (type 5), or identity class (type 6), this field is set to 0. See parameter UEPMRTYP.

UEPFCLNO
Address of the fullword number of field connectors. This field has meaning only for performance class records. If the monitoring record type is exception class (type 4), transaction resource (type 5), or identity class (type 6), this field is set to 0. See parameter UEPMRTYP.

UEPMRTYP
Address of the halfword monitoring record type. The monitoring record type value can be one of the numbers shown in the following table:

Table 9. Monitoring record type values and their meanings

Record type value	Meaning
3	Performance class monitoring record
4	Exception class monitoring record
5	Transaction resource monitoring record

Table 9. Monitoring record type values and their meanings (continued)

Record type value	Meaning
6	Identity class monitoring record

UEPMRLEN
Address of the fullword monitoring record length.

UEPMREC
Address of the monitoring record. The length of UEPMREC is addressed by the parameter UEPMRLEN.

UEPSRCTK
Address of the z/OS Workload Manager service reporting class token for the current transaction. If CICS support for z/OS Workload Manager is not available, this token is null.

UEPMPREC
Address of the monitoring performance record. This field has meaning only for performance class records. If the monitoring record type is exception class (type 4), transaction resource (type 5), or identity class (type 6), this field is set to 0. See parameter UEPMRTYP. The performance record addressed by this parameter must be mapped using the DFHMNTDS DSECT, and must not be mapped using the UEPDICT and UEPDICTE dictionary parameters.

Return codes

UERCNORM
Continue processing.

UERCBYP
Suppress monitor record output.

UERCPURG
Task purged during XPI call.

XPI calls
WAIT_MVS can be used. **Do not use any other calls**.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

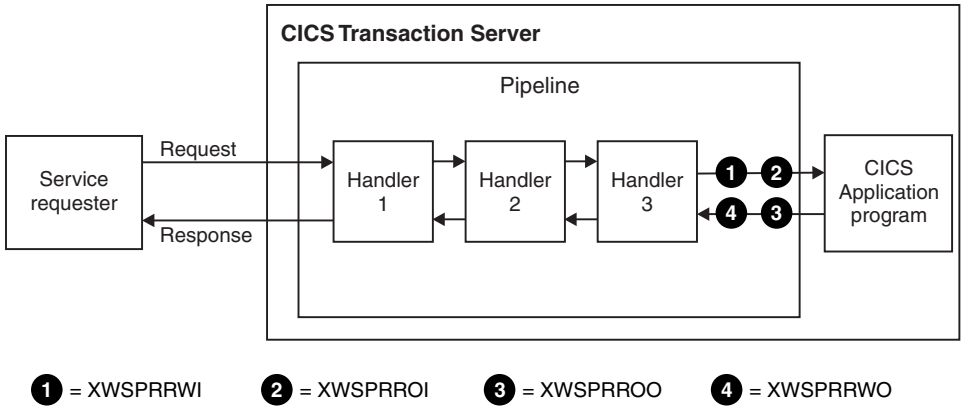
"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Pipeline domain exits

Use the pipeline domain exits to customize the processing that occurs for inbound and outbound Web services in the pipeline. You can use the pipeline domain exits to access containers on a Web services provider pipeline, a Web services requester pipeline, or a Web services requester pipeline that contains a security message handler.

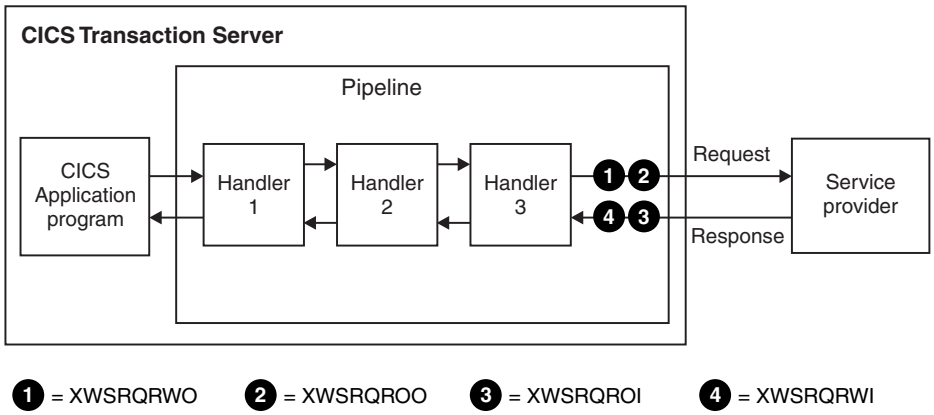
GLUE points in the provider pipeline

Global user exit (GLUE) points that you can use in a provider pipeline, or a secured provider pipeline, have a prefix of XWSPR. This diagram shows the order in which the GLUE points can be used:

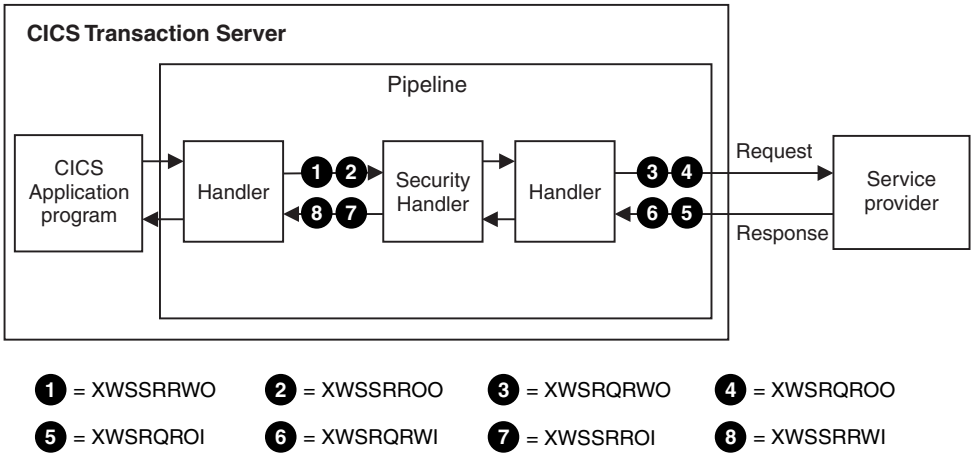


GLUE points in the requester pipeline

GLUE points that you can use in a requester pipeline have a prefix of XWSRQ. This diagram shows the order in which the GLUE points can be used:



GLUE points that you can use in a secured requester pipeline have a prefix of XWSSR. There are eight GLUE points that can be used in a pipeline containing a security handler; four of these can be used only in a secure requester pipeline and four can be used in any requester pipeline. This diagram shows the order in which the GLUE points can be used:



Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XWSPRRWI

Use the XWSPRRWI exit to access containers on the current channel that are to be processed by the Web services provider application, after CICS has converted the Web services request body into a language structure and before any instance of the XWSPRROI exit is invoked.

You can use this exit to issue API and SPI commands to examine and update any information in the containers and to issue a SOAP fault.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
Address of the 4-byte terminal ID. The value is null for a Web service provider.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

Return codes

UERCNORM
Continue processing.

UERCRPIP
Do not continue on the pipeline.

XPI calls

No XPI interfaces are available.

API and SPI commands

The following commands are supported:

- **EXEC CICS DELETE CONTAINER**
- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**
- **EXEC CICS PUT CONTAINER**
- **EXEC CICS SOAPFAULT ADD**
- **EXEC CICS SOAPFAULT CREATE**
- **EXEC CICS SOAPFAULT DELETE**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XWSPRROI

Use the XWSPRROI exit to access containers on the current channel before the containers are processed by a Web services provider application, but after any instance of the XWSPRRWI exit is invoked.

You can use this exit to issue API and SPI commands to examine any information that is processed by the Web services business application. You cannot issue a SOAP fault or update any of the information. CICS ignores any return code specified in register 15 after the global user exit program has finished.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
Address of the 4-byte terminal ID. The value is null for a Web service provider.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

Return codes

UERCNORM
Continue processing.

XPI calls

No XPI interfaces are available.

API and SPI commands

You can use the following commands:

- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XWSPRROO

Use the XWSPRROO exit to access containers on the current channel after the Web services provider application issues the Web service response message and before CICS creates the body of the response message.

You can use this exit to issue API and SPI commands to examine the containers on the current channel. You cannot issue a SOAP fault or update any of the containers. CICS ignores any return code specified in register 15 after the global user exit program has finished.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
Address of the 4-byte terminal ID. The value is null for a Web service provider.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

UEPAPAB
A 1-byte field indicating whether the Web service provider application completed its processing successfully. Valid values are as follows:

UEPAPABY (X'80')
The Web service provider application abended.

UEPAPABN (X'40')
The Web service provider application completed its processing successfully.

UEPAPSF
A 1-byte field indicating whether the Web service provider application set a SOAP fault. Valid values are as follows:

UEPAPSFY (X'80')
The Web service provider application is returning a SOAP fault.

UEPAPSFN (X'40')
The Web service provider application is not returning a SOAP fault.

Return codes

UERCNORM
Continue processing.

XPI calls

No XPI interfaces are available.

API and SPI commands

The following commands are supported:

- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Related reference:

"Pipeline sample exit program (DFH$PIEX)" on page 30
CICS includes a sample global user exit program that provides sample processing for the XWSPRROO global user exit. The sample program is DFH$PIEX.

Exit XWSPRRWO

Use the XWSPRRWO exit to access containers on the current channel that have been processed by a Web services provider application after any instance of the XWSPRROO exit.

You can use this exit to issue API and SPI commands to examine and update any information in the containers and to issue a SOAP fault. For example, you can add additional SOAP headers to an outbound SOAP response. Any updates to the current channel container data are processed by CICS and returned to the requester. CICS ignores any return code specified in register 15 after the global user exit program has finished.

Exit-specific parameters

UEPTRANID
: Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
: Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
: Address of the 4-byte terminal ID. The value is null for a Web service provider.

UEPPROG
: Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
: Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

UEPAPAB
A 1-byte field indicating whether the Web service provider application completed its processing successfully. Valid values are as follows:

UEPAPABY (X'80')
The Web service provider application abended.

UEPAPABN (X'40')
The Web service provider application completed its processing successfully.

UEPAPSF
A 1-byte field indicating whether the Web service provider application set a SOAP fault. Valid values are as follows:

UEPAPSFY (X'80')
The Web service provider application is returning a SOAP fault.

UEPAPSFN (X'40')
The Web service provider application is not returning a SOAP fault.

Return codes

UERCNORM
Continue processing.

XPI calls

No XPI interfaces are available.

API and SPI commands

You can use the following commands:

- **EXEC CICS DELETE CONTAINER**
- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**
- **EXEC CICS PUT CONTAINER**
- **EXEC CICS SOAPFAULT ADD**
- **EXEC CICS SOAPFAULT CREATE**
- **EXEC CICS SOAPFAULT DELETE**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XWSRQRWO

Use the XWSRQRWO exit to access containers on the current channel before they are passed to the transport to be processed. This exit runs after CICS has converted the application's language structure into a Web services request body and before CICS processes the optional XWSRQROO exit point.

You can use this exit to issue API and SPI commands to examine and update any information in the containers on the current channel. This information is available to any instance of the XWSRQRWO exit and also to the outbound Web services provider. You cannot issue a SOAP fault. A return code can be put in register 15 to indicate that CICS does not continue the pipeline after the global user exit program finishes.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program which issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

Return codes

UERCNORM
Continue processing.

UERCRPIP
Do not continue on the pipeline.

XPI calls

No XPI interfaces are available.

API and SPI commands

You can use the following commands:

- **EXEC CICS DELETE CONTAINER**
- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**
- **EXEC CICS PUT CONTAINER**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XWSRQROO

Use the XWSRQROO exit to access containers on the current channel before they are passed to the transport to be processed. This exit runs after any instance of the XWSRQRWO exit is processed and before the data flowing outbound on the Web services transport.

You can use this exit to issue API and SPI commands to examine any information in the containers. This information is processed by the outbound Web services provider. You cannot issue a SOAP fault. CICS ignores any return code specified in register 15 after the global user exit program has finished.

Exit-specific parameters

UEPTRANID
: Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
: Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
: Address of the 4-byte terminal ID.

UEPPROG
: Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
: Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
: Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

Return codes

UERCNORM
Continue processing.

UERCRPIP
Do not continue on the pipeline.

XPI calls

No XPI interfaces are available.

API and SPI commands

You can use the following commands:

- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XWSRQROI

Use the XWSRQROI exit to access containers on the current channel after they are processed by the transport as a Web services response. The XWSRQROI exit is invoked directly after CICS has processed the outbound Web service provider. It can also be invoked before any instance of the XWSRQRWI exit.

You can use this exit to issue API and SPI commands to examine any information in the containers. This information is processed by the outbound Web services provider. You cannot issue a SOAP fault. CICS ignores any return code specified in register 15 after the global user exit program has finished.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

Return codes

UERCNORM
Continue processing.

XPI calls

No XPI interfaces are available.

API and SPI commands

You can use the following commands:

- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XWSRQRWI

Use the XWSRQRWI exit to access containers on the current channel after they have been processed by the transport as a Web services response. The XWSRQRWI exit is invoked directly after CICS has processed the inbound Web service response. It is also invoked after any instance of the XWSRQROI exit.

You can use this exit to issue API and SPI commands to examine and update any information in the containers. This information is processed by the outbound Web services provider and is received by the associated Web services requester application. You cannot issue a SOAP fault. CICS ignores any return code specified in register 15 after the global user exit program has finished.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

Return codes

UERCNORM
Continue processing.

XPI calls

No XPI interfaces are available.

API and SPI commands

You can use the following commands:

- **EXEC CICS DELETE CONTAINER**
- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**
- **EXEC CICS PUT CONTAINER**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XWSSRRWO

Use the XWSSRRWO exit to access containers on the current channel, with CICS acting as a secured Web services requester, before they are passed to the transport to be processed. This exit runs after CICS converts the application's language structure into a Web services request body and before CICS processes the optional XWSSRROO exit point, and before being encrypted by the pipeline's security handler.

You can use this exit to issue API and SPI commands to examine and update any information in the containers. This information is available to any instance of the XWSSRRWO exit and also to the outbound Web services provider. You cannot

issue a SOAP fault. A return code can be specified in register 15 to indicate that CICS does not continue the pipeline after the global user exit program finishes.

If the pipeline does not contain a security handler, this exit is not driven. See the "Exit XWSRQRWO" on page 221 topic for instances of the pipeline not containing a security handler.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

Return codes

UERCNORM
Continue processing.

UERCRPIP
Do not continue on the pipeline.

XPI calls

No XPI interfaces are available.

API and SPI commands

You can use the following commands:

- **EXEC CICS DELETE CONTAINER**
- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**
- **EXEC CICS PUT CONTAINER**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XWSSRROO

Use the XWSSRROO exit to access containers on the current channel, with CICS acting as a secured Web services requester, before they are passed to the transport to be processed. This exit runs after any instance of the XWSSRRWO exit is processed and before the encryption of data flowing outbound on the Web services transport.

If the pipeline does not contain a security handler, this exit is not driven. See the "Exit XWSRQROO" on page 222 topic for instances of the pipeline not containing a security handler

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

Return codes

UERCNORM
Continue processing.

XPI calls

No XPI interfaces are available.

API and SPI commands

You can use the following commands:

- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XWSSRROI

Use the XWSRQROI exit to access containers on the current channel, with CICS acting as a secured Web services requester, after they are processed by the transport as a Web services response. This exit runs after CICS processes the Web service response and before any instance of the XWSSRRWI exit.

You can use this exit to issue API and SPI commands to examine any information in the containers. This information is processed by the outbound Web services provider. You cannot issue a SOAP fault. CICS ignores any return code specified in register 15 after the global user exit program has finished.

If the pipeline does not contain a security handler, this exit is not driven. See the "Exit XWSRQROI" on page 223 topic for instances of the pipeline not containing a security handler.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

Return codes

UERCNORM
Continue processing.

XPI calls

No XPI interfaces are available.

API and SPI commands

You can use the following commands:

- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XWSSRRWI

Use the XWSSRRWI exit to access containers on the current channel, with CICS acting as a secured Web services requester, after they have been processed by the transport as a Web services response. This exit runs after CICS processes the Web service response and after any instance of the XWSSRROI exit.

You can use this exit to issue API and SPI commands to examine and update any information in the containers. This information is processed by the Web services requester application. You cannot issue a SOAP fault. CICS ignores any return code specified in register 15 after the global user exit program has finished.

If the pipeline does not contain a security handler, this exit is not driven. See the "Exit XWSRQRWI" on page 224 topic for instances of the pipeline not containing a security handler.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID. The exit or task runs under this transaction ID.

UEPUSER
Address of the 8-byte user ID. The user ID is the one associated with the transaction ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name. The application program name is that of the program that issued the **INVOKE SERVICE** or **INVOKE WEBSERVICE** command.

UEPCHANN
Address of a 16-byte field that contains the name of the current channel. You can identify the channel explicitly using this parameter.

UEPCONTR
Address of a 16-byte field that contains the name of the data container on the channel named by UEPCHANN. This container holds the application data structure.

Return codes

UERCNORM
Continue processing.

XPI calls

No XPI interfaces are available.

API and SPI commands

You can use the following commands:

- **EXEC CICS DELETE CONTAINER**
- **EXEC CICS GET CONTAINER**
- **EXEC CICS INQUIRE WEBSERVICE**
- **EXEC CICS PUT CONTAINER**

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Program control program exits (XPCREQ, XPCERES, XPCREQC, XPCFTCH, XPCHAIR, XPCTA, and XPCABND)

These exits are invoked before or after CICS program control operations, including program link requests, a program receiving control, and a transaction abend.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Program control exits XPCREQ, XPCERES, XPCREQC

These exits are called by the EXEC interface program before a link request is processed, before CICS processes dynamically routed link requests, or after a link request has completed.

XPCREQ
XPCREQ is called by the EXEC interface program before a link request is processed. If the request is a distributed program link, the XPCREQ exit is driven on both sides of the link; that is, in both the client and the server regions. The exit program is passed the address of the application's parameter list (in UEPCLPS), and can modify this list as required. For example, you can use this exit to modify the SYSID at the time of a distributed program link

request. You can write an application program to manage a list of SYSIDs in a global work area (GWA). The global user exit program can obtain access to the GWA, and use the information stored there to redirect DPL requests.

Note:

1. The attributes of the local PROGRAM resource are not passed to the exit program. If the exit program requires the value of an attribute, it can issue an **EXEC CICS INQUIRE PROGRAM** command.
2. If you use XPCREQ to change the target SYSID, remember that:
   a. If SYSID specifies a remote region, no reference is made to the local PROGRAM resource. In the remote region the program runs under the TRANSID of the transaction in the client region, not under the TRANSID specified on the PROGRAM resource in the client region.
   b. If SYSID specifies the local region, CICS treats the link request as if SYSID was not specified. The local PROGRAM resource is honored.
   c. The XPCREQ exit is called by internal requests made by CICS code, in addition to requests made by applications.

XPCERES

XPCERES is called by the EXEC interface program before CICS processes either of the following kinds of dynamically routed link request:

- A distributed program link (DPL) call
- A Link3270 bridge request

XPCERES is called:

- After exit XPCREQ and before XPCREQC if these exits are enabled:
  - If an XPCREQ exit program chooses to bypass the request, XPCERES is not called.
  - If an XPCREQ exit program modifies the command parameter list, XPCERES must deal with the modified request.
- On the target region to which the request has been routed.
- Only if it is enabled. Enable this exit only in application-owning regions when DPL and Link3270 bridge requests can be dynamically routed.
- By internal requests made by CICS code, in addition to requests made by applications.

The XPCERES exit is not called:

- For statically routed requests.
- If it is disabled.
- If an XPCREQ exit program chooses to bypass the request.

You can use XPCERES to check that all resources required by the linked-to program are available on the target region. If the program is disabled or a required file is missing, your exit program can give the dynamic routing program the opportunity to route the request to a different region. Set a return code of UERCRESU. CICS performs the following processing:

1. In the COMMAREA of the routing program, CICS sets the DYRERROR field to 'F' - resource unavailable.
2. CICS calls the routing program, on the routing region, for route selection failure.
3. CICS returns a RESUNAVAIL condition on the EXEC CICS LINK command that was run by the mirror on the target region. This condition is not returned to the application program.

CICS ignores any changes made by the exit program to the values of any of the exit parameters. Your exit program can set a return code, but not change any parameters.

For guidance information about dynamically routing DPL requests, see Dynamically routing DPL requests. For guidance information about dynamically routing Link3270 bridge requests, see Using Link3270 bridge load balancing in Administering. For programming information about writing a dynamic routing program to route DPL requests, see "Routing DPL requests dynamically" on page 583. For programming information about writing a dynamic routing program to route Link3270 bridge requests, see "Routing bridge requests dynamically" on page 589.

XPCREQC

XPCREQC is called after a link request has completed. You can use this exit to pass back a response to the application by using the EIBRESP or EIBRESP2 fields. Such responses might be used to keep status information about a link request up-to-date. For example, if a link request fails because a connection is unavailable, XPCREQC can set EIBRESP=500 (a response code not used by CICS) to indicate the failure, enabling the application, with the other exit XPCREQ, to determine a suitable course of action.

Note: The XPCREQC exit is called by internal requests made by CICS code, in addition to requests made by applications.

Exit XPCREQ:

Exit XPCREQ is invoked by the EXEC interface program before a link request is processed.

When invoked

By the EXEC interface program before a link request is processed.

Exit-specific parameters

UEPCLPS

Address of the command parameter list.

UEPPCTOK

Address of a 4-byte token to be passed to XPCREQC. This allows you, for example, to pass a work area to exit XPCREQC.

UEPRCODE

Address of a 6-byte hexadecimal copy of EIBRCODE.

UEPRECUR

Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPRESP

Address of a 4-byte copy of EIBRESP.

UEPRESP2

Address of a 4-byte copy of EIBRESP2.

UEPTSTOK

Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

UEP_PC_PBTOK
Address of a 4-byte field containing the z/OS Workload Manager (WLM) Performance Block Token. An exit program can use this token to access information (such as the service class token, SERVCLS) in the WLM Performance Block. To do so, it must use the WLM EXTRACT macro, IWMMEXTR, passing the Performance Block Token as the MONTKN input parameter. For more information about the IWMMEXTR macro, see *z/OS MVS Programming: Workload Management Services*.

An exit program must not attempt to modify the Performance Block: if it does so, the results are unpredictable.

Return codes

UERCBYP
Program control is to ignore the request.

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.

API and SPI calls
All can be used, except for:
EXEC CICS SHUTDOWN
EXEC CICS XCTL

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XPCERES:

Exit XPCERES is invoked by the EXEC interface program, before processing of a program link or Link3270 bridge request that has been dynamically routed to this region, where the routing region supports the "resource unavailable" condition.

Exit-specific parameters

Note: CICS ignores any changes made by the exit program to the values of any of the exit parameters. Your exit program can set a return code, but not change any parameters.

UEPCLPS
Address of the command parameter list.

UEPPCTOK
Address of a 4-byte token to be passed to XPCREQC.

UEPRCODE
Address of a 6-byte hexadecimal copy of EIBRCODE.

UEPRECUR
Address of a halfword recursion counter. Because the XPCERES exit can never be called recursively in the same transaction, the value of this field is always 0.

UEPRESP
Address of a 4-byte copy of EIBRESP.

UEPRESP2
Address of a 4-byte copy of EIBRESP2.

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

UEP_PC_PBTOK
Address of a 4-byte field containing the z/OS Workload Manager (WLM) Performance Block Token. An exit program can use this token to access information (such as the service class token, SERVCLS) in the WLM Performance Block. To do so, it must use the WLM EXTRACT macro, IWMMEXTR, passing the Performance Block Token as the MONTKN input parameter. For more information about the IWMMEXTR macro, see *z/OS MVS Programming: Workload Management Services*.

An exit program must not attempt to modify the Performance Block: if it does so, the results are unpredictable.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

UERCRESU
A required resource is unavailable. Setting this value causes CICS to reject the routed request, and to return a value of 'F' (resource unavailable) in the DYRERROR field of the routing program's communications area.

XPI calls
All can be used.

Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.

API and SPI calls
All except EXEC CICS SHUTDOWN and EXEC CICS XCTL can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XPCREQC:

Exit XPCREQC is invoked on completion of a program control link request.

When invoked
On completion of a program control link request.

Exit-specific parameters

UEPCLPS
Address of the command parameter list.

UEPPCTOK
Address of a 4-byte token passed from XPCREQ. This allows XPCREQ to, for example, pass a work area to XPCREQC.

UEPRCODE
Address of a 6-byte hexadecimal copy of EIBRCODE.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPRESP
Address of a 4-byte copy of EIBRESP.

UEPRESP2
Address of a 4-byte copy of EIBRESP2.

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

UEP_PC_REMOTE_SYSTEM
If the request is to be sent to a remote region, is the address of an area containing the 4-byte name of the remote region. (The remote region may have been specified by, for example, the SYSID option

of the EXEC CICS LINK command, function shipping, work-load management, or the REMOTESYSTEM option of the PROGRAM definition.)

If the request is to be executed on the local region, this parameter is the address of a 4-byte area containing blanks.

UEP_PC_REMOTE_NAME
If the program is to be executed in a remote system, is the address of an area containing the name of the program, as it is known in the remote system.

UEP_PC_PBTOK
Address of a 4-byte field containing the z/OS Workload Manager (WLM) Performance Block Token. An exit program can use this token to access information (such as the service class token, SERVCLS) in the WLM Performance Block. To do so, it must use the WLM EXTRACT macro, IWMMEXTR, passing the Performance Block Token as the MONTKN input parameter. For more information about the IWMMEXTR macro, see *z/OS MVS Programming: Workload Management Services*.

An exit program must not attempt to modify the Performance Block: if it does so, the results are unpredictable.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.

API and SPI calls
All can be used, except for:
EXEC CICS SHUTDOWN
EXEC CICS XCTL

Note: Take care when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a program control request is issued from the XPCREQ or XPCREQC exits.

Use of the recursion counter UEPRECUR is recommended.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The command parameter structure:

The command parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of a bit string that describes the type of request and identifies each keyword specified with the request.

The remaining addresses point to pieces of data associated with the request; for instance, the second address always points to the program name. You can examine the parameters in the list to determine the values of the keywords. You can also modify values of parameters specified on the request. For example, you could change the name of the program involved in the request, or add the SYSID to route the link request to a remote system.

End of parameter list indicator

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

For example, if the parameter list specifies only the first two addresses (PC_ADDR0, the address of the EID, and PC_ADDR1, the address of the name of the program named in the link request), the high-order bit is set on in PC_ADDR1. If you extend the parameter list by setting the address of a SYSID in PC_ADDR7, you must unset the high-order bit in PC_ADDR1 and set it on in PC_ADDR7 instead.

The original parameter list, as it was before XPCREQ was invoked, is restored after the completion of XPCREQC. It follows that EDF will display the original command before **and** after execution: **EDF will not display any changes made by the exit**.

The UEPCLPS exit-specific parameter:

The UEPCLPS exit-specific parameter is the address of the command-level parameter structure, and is included in exits XPCREQ and XPCREQC.

The command-level parameter structure contains 11 addresses, PC_ADDR0 through PC_ADDRA. It is defined in the DSECT PC_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHPCEDS.

The command-level parameter list is made up as follows:

PC_ADDR0
is the address of a 7-byte area called the EXEC interface descriptor (EID), which is made up as follows:

- **PC_GROUP**
- **PC_FUNCT**
- **PC_BITS1**
- **PC_BITS2**
- **PC_EIDOPT4**
- **PC_EIDOPT5**
- **PC_EIDOPT6**

PC_GROUP
Always X'0E', indicating that this is a program control request.

PC_FUNCT
One byte which defines the type of request, which for XPCREQ and XPCREQC is always X'02', indicating a LINK request.

PC_BITS1
Existence bits that define which keywords that contain values were specified. To obtain the value associated with a keyword, you need to use the appropriate address from the command-level parameter list. Before using this address you must check the associated existence bit to ensure that the address is valid. If the existence bit is set off, the keyword was not specified in the request and the address should not be used. The symbolic and hexadecimal values of the existence bits are as follows:

PC_EXIST1 (X'80 ')
Set if the request contains the keyword PROGRAM. If set, **PC_ADDR1** is meaningful. (This bit should always be set for a LINK request.)

PC_EXIST2 (X'40 ')
Set if the request specifies the COMMAREA parameter. If set, **PC_ADDR2** is meaningful.

PC_EXIST3 (X'20 ')
Set if the request specifies the LENGTH parameter. If set, **PC_ADDR3** is meaningful.

PC_EXIST4 (X'10 ')
Set if the request specifies the INPUTMSG parameter. If set, **PC_ADDR4** is meaningful.

PC_EXIST5 (X'08 ')
Set if the request specifies the INPUTMSGLEN parameter. If set, **PC_ADDR5** is meaningful.

PC_EXIST6 (X'04 ')
Set if the request specifies the DATALENGTH parameter. If set, **PC_ADDR6** is meaningful.

PC_EXIST7 (X'02 ')
Set if the request specifies the SYSID parameter. If set, **PC_ADDR7** is meaningful.

PC_EXIST8 (X'01 ')
Set if the request specifies the TRANSID parameter. If set, **PC_ADDR8** is meaningful.

PC_BITS2
One byte containing one of the following values:

PC_EXIST9 (X'80')
Not used.

PC_EXISTA (X'40')
Set if the request specifies the CHANNEL parameter. If set, PC_ADDRA is meaningful.

PC_EIDOPT4
Not used by program control.

PC_EIDOPT5
Not used by program control.

PC_EIDOPT6
Indicates whether the request specifies the SYNCONRETURN option. If it does, X'80' is set.

PC_ADDR1
is the address of an 8-byte area containing the program name from the PROGRAM parameter.

PC_ADDR2
is the address of the COMMAREA data.

PC_ADDR3
is the address of a 2-byte area containing the length of the COMMAREA, as a half-word binary value.

PC_ADDR4
is the address of the INPUTMSG data.

PC_ADDR5
is the address of a 2-byte area containing the length of the INPUTMSG, as a half-word binary value.

PC_ADDR6
is the address of a 2-byte area containing the length specified on the DATALENGTH parameter, defining how much data is to be sent from the COMMAREA. The length is held as a half-word binary value.

PC_ADDR7
is the address of the 4-byte name of the remote system the LINK request is to be shipped to, as specified on the SYSID parameter.

PC_ADDR8
is the address of the 4-byte name of the mirror transaction to be attached in the remote system, as specified on the TRANSID parameter.

PC_ADDR9
is not used.

PC_ADDRA
is the address of the 16-byte channel name, as specified on the CHANNEL parameter.

Modifying fields in the command parameter structure:

Some fields that are passed to program control are used as input to the request, some are used as output fields, and some are used for both input and output. The method your user exit program uses to modify a field depends on the usage of the field.

Modifying input fields:

The correct method of modifying an input field is to create a new copy of it, and to change the address in the command parameter list to point to your new data.

Note: You must never modify an input field by altering the data that is pointed to by the command parameter list. To do so would corrupt storage belonging to the application program and could cause a failure when the program attempted to reuse the field.

Modifying output fields: The technique described in "Modifying input fields" on page 239 is not suitable for modifying output fields. (The results would be returned to the new area instead of the application's area, and would be invisible to the application.)

An output field is modified by altering the data that is pointed to by the command-level parameter list. In the case of an output field you can modify the application's data in place, because the application is expecting the field to be modified anyway.

Modifying the EID:

It is not possible to modify the EID to make major changes to requests. It is not possible, for example, to change a LINK request to a different type of Program Control request. However, you can make minor changes to requests, such as to turn on the existence bit for SYSID so that the request can be changed into one that is shipped to a remote system.

The list that follows shows the bits in the EID that can be modified. Any attempt to modify any other part of the EID is ignored.

PC_BITS1

X'40' The existence bit for the COMMAREA

X'20' The existence bit for LENGTH

X'10' The existence bit for INPUTMSG

X'08' The existence bit for INPUTMSGLEN

X'04' The existence bit for DATALENGTH

X'02' The existence bit for SYSID

X'01' The existence bit for TRANSID.

PC_BITS2

X'40' The existence bit for CHANNEL.

PC_EIDOPT5

Not used for a PC link request.

Bits in the EID should be modified in place. You should not modify the pointer to the EID. (Any attempt to do so is ignored by CICS.)

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the program control request only.

Your user exit program is prevented from making major changes to the EID.

Using the program control request token, UEPPCTOK:

UEPPCTOK provides the address of a 4-byte area that you can use to pass information between the XPCREQ and XPCREQC user exits for the same program control request.

For example, the address of a piece of storage that is obtained by the XPCREQ user exit, which has to be freed by the XPCREQC user exit, can be passed in the UEPPCTOK field.

UEPPCTOK is usable only for the duration of a single program control request, because its contents might be destroyed at the end of the request. If you need to pass information between successive invocations of a global user exit, you can use task token UEPTSTOK to do so. For more information about UEPTSTOK, see "Using the task token UEPTSTOK" on page 19.

The EIB:
Copies of EIBRSRCE, EIBRCODE, EIBRESP, and EIBRESP2 are passed to the exit, so that you can:

- Modify or set completion or resource information in XPCREQ and XPCREQC.
- Examine completion information in XPCREQC.

You can update the copies of EIBRSRCE, EIBRCODE, EIBRESP and EIBRESP2 that you are given in the parameter list. Program Control copies your values into the real EIB after the completion of XPCREQC; or if you specify a return code of 'bypass' in XPCREQ.

You must set valid program control responses. You must set all three of EIBRCODE, EIBRESP and EIBRESP2 to a consistent set of values, such as would be set by Program Control to describe a valid completion. **Program Control does not police the consistency of EIBRCODE, EIBRESP, and EIBRESP2**. To aid you in setting the values of EIBRCODE, EIBRESP, and EIBRESP2, the values used by Program Control are specified in DFHPCEDS.

Example of how XPCREQ and XPCREQC can be used:

In this example, XPCREQ and XPCREQC are used to route LINK requests to a number of different CICS regions to provide a simple load balancing mechanism.

The example shows only the capabilities of the exits; it is not intended to indicate an ideal way of achieving the load balancing function. For the purpose of this example, it is assumed that a global work area (GWA) already exists, and that it contains a list of available SYSIDs together with a count of the number of LINK requests currently being processed by each SYSID.

In XPCREQ:

1. Scan the global work area (GWA) to locate a suitable CICS region; for example, the region currently processing the least number of LINK requests.
2. Having decided which system to route the request to, increment the use count for this system.
3. Obtain a 4-byte area in which to store the SYSID for this request (this can be allocated from the GWA to avoid issuing a GETMAIN). If the area is obtained by issuing a GETMAIN, set UEPPCTOK to the address of the storage obtained.
4. Set PC_ADDR7 to the address of the 4-byte area.
5. If setting PC_ADDR7 now makes it the last address, set the high-order bit in the address, and unset the high-order bit in what was previously the last address.
6. Set the X'02' existence bit on in PC_BITS1 to indicate that a SYSID is specified.
7. Return to CICS.

In XPCREQC:

1. Scan the global work area (GWA) and locate the entry for the CICS region specified in the SYSID parameter.
2. Decrement the use count for this system.

3. If a GETMAIN was issued in XPCREQ to obtain an area to hold the SYSID, issue a FREEMAIN for the address held in UEPPCTOK.
4. Return to CICS.

Exit XPCFTCH

XPCFTCH is invoked before a program defined to CICS (including internal CICS modules) receives control, which could be because it is the first program in a transaction, or as a result of a LINK, XCTL, or HANDLE ABEND PROGRAM request.

You can use this exit to modify the entry address used when linking to the program. If the exit sets a return code of zero, or a modified address of zero, the entry address of the original application program is used.

You use this exit to pass control to an AMODE(64), AMODE(31), or AMODE(24) assembler application program or routine before the original program is invoked. After this assembler program finishes its processing, it should pass control back to the entry point of the original program by using a branch instruction. Do not use the exit to invoke any program other than the original program, because the results are unpredictable.

When XPCFTCH is invoked for a C or C++ program that is compiled with the XPLINK option, a flag is set to ignore any modification of the entry point address that the user exit might make.

If a modified entry address is supplied, the program that is invoked receives control in the execution key that the original application program would have received control in; that is, as specified on the EXECKEY option of the resource definition of the original program.

When invoked

Before an application program receives control.

Exit-specific parameters

UEPPCDS

Address of a storage area that contains program- and terminal-related information, and that can be mapped using the DSECT DFHPCUE. When XPCFTCH is invoked, the following DFHPCUE fields are significant:

PCUE_CONTROL_BITS

- 1-byte flag field. A setting of PCUECBTE indicates that the transaction is linked to a terminal.
- A setting of PCUENOTX (X'40') indicates that the program is not command level.
- A flag, PCUE_NO_MODIFY, in PCUE_CONTROL_BITS indicates that a modified entry address is not supported. When set, any return code of UERCMEA from XPCFTCH is ignored. CICS sets this flag before invoking XPCFTCH for C and C++ programs compiled with the XPLINK option.
- A setting of PCUE_REAL (X'20') indicates that a real entry point is set in PCUE_REAL_ENTRY.

PCUE_TASK_NUMBER

3-character packed decimal field that contains the task number.

PCUE_TRANSACTION_ID
4-character field that contains the ID of the original transaction. This ID might differ from the current transaction ID.

PCUE_TERMINAL_ID
4-character field that contains the terminal ID (if any).

PCUE_PROGRAM_NAME
8-character field that contains the name of the program that is to receive control.

PCUE_PROGRAM_LANGUAGE
3-character field that contains the language of the program that is to receive control.

PCUE_LOAD_POINT
The load point of the program.

PCUE_ENTRY_POINT
The entry point of the program.

PCUE_AMOD
The addressing mode of the program is AMODE(31). This field is provided for compatibility with existing exit programs.

PCUE_AMOD_31
The addressing mode of the program is AMODE(31). Use this field in preference to PCUE_AMOD.

PCUE_AMOD_64
The addressing mode of the program is AMODE(64).

PCUE_PROGRAM_SIZE
Fullword that contains the size of the program, in bytes.

PCUE_COMMAREA_ADDRESS
Address of the communication area of the program, if the program has one.

PCUE_COMMAREA_SIZE
Fullword that contains the length of the communication area of the program, if the program has one.

PCUE_LOGICAL_LEVEL
Fullword that contains the program logical level.

PCUE_BRANCH_ADDRESS
Fullword. Use this field to supply an alternative entry address. Set the top bit to specify that the alternative program is to run AMODE (31).

PCUE_REAL_ENTRY
From z/OS 1.7 onwards, this field provides the real entry point for Language Environment conforming programs. Previously, only PCUE_ENTRY_POINT was available to you, but for Language Environment conforming programs, this field did not contain the entry point that you needed to know about.

Note: With z/OS 1.7, this field provides a solution to the problem raised by APAR PQ43992.

PCUE_CHANNEL_NAME
Address of a 16-byte field that contains the name of the channel

with which the application program is to be invoked (that is, the current channel of the program). If there is no channel, this field is set to blanks.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

UERCMEA
Entry address has been modified.

XPI calls

All can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The sample XPCFTCH global user exit program, DFH$PCEX

A CICS-supplied sample exit program, DFH$PCEX, is designed to be driven by the XPCFTCH exit.

For more information about DFH$PCEX, see "Sample global user exit programs" on page 19.

Exit XPCHAIR

Exit XPCHAIR is invoked before a HANDLE ABEND LABEL routine is given control.

This situation occurs only when a program abend causes a branch to an internal abend routine. When the HANDLE ABEND request specifies PROGRAM, exit XPCFTCH is invoked (see "Exit XPCFTCH" on page 242). You can use the XPCHAIR exit to supply an alternative handle abend address. If the exit sets a return code of zero, or an alternative address of zero, CICS passes control to the specified internal routine of the application program

This exit is never invoked for a C or C++ program that is compiled with the XPLINK option, or for an AMODE(64) program.

If a modified entry address is supplied:

- The code that is invoked receives control in the execution key that the internal abend routine would have received control in; that is, the key in force when the EXEC CICS HANDLE ABEND LABEL command was issued.
- The resume address is placed in register 14 for a COBOL application or register 15 for an Assembler application. If your application runs in a mixed

environment, your exit program might need to set up its own base register. For example, you could use the following code to set up addressability:

```
BASSM 15,0
USING *,15
```

When invoked

Before a HANDLE ABEND routine is given control.

Exit-specific parameters

UEPPCDS

Address of a storage area that contains program and terminal related information, and that can be mapped using the DSECT DFHPCUE. When XPCHAIR is invoked, the following DFHPCUE fields are significant:

PCUE_CONTROL_BITS

1-byte flag field. A setting of PCUECBTE indicates that the transaction is linked to a terminal.

PCUE_TASK_NUMBER

3-character packed decimal field that contains the task number.

PCUE_TRANSACTION_ID

4-character field that contains the transaction ID.

PCUE_TERMINAL_ID

4-character field that contains the terminal ID (if any).

PCUE_PROGRAM_NAME

8-character field that contains the name of the program that issued the HANDLE ABEND LABEL command.

PCUE_LOGICAL_LEVEL

Fullword that contains the program logical level.

PCUE_BRANCH_ADDRESS

Fullword. Use this field to supply the address of an alternate abend routine. Set the top bit to specify that the alternate abend routine is to run AMODE (31).

UEPTACB

Address of the transaction abend control block (TACB) for the abend. If the abend occurred because of a program check, the information in the TACB includes:

- The program status word (PSW).
- The registers at the time of the abend.
- Details of the subspace and access registers current at the time of the abend.
- The Breaking Event Address Register (BEAR).

You can map the TACB using the DFHTACB TYPE=DSECT macro.

Return codes

UERCNORM

Continue processing.

UERCPURG

Task purged during XPI call.

UERCMEA
The address of an alternate abend routine is supplied.

XPI calls
All can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XPCTA

Exit XPCTA is invoked immediately after a transaction abend, and before any processing that might modify the existing environment so that the task could not be resumed.

You can use the XPCTA exit to do the following:

- Set a resume address, instead of letting CICS process the abend
- Specify the subspace that control is passed in

If a resume address is passed back, registers 0 through 13 and 15 are restored to their values at the time of the abend. Register 14 is used to branch to the resume address. If the exit sets a return code of zero, or a resume address of zero, CICS processes the abend.

If the transaction abend occurs as a result of a program check or an operating system abend, the XDUREQ dump domain exit might be invoked before XPCTA (see "Exit XDUREQ" on page 68). Also, if a resume address is passed back, registers 0 through 15 are restored to their value at the time of the abend. The program status word (PSW) is used to branch to the resume address.

In some situations, CICS sets a flag to ignore the resume address that is usually obtained from the UERCMEA return code.

- CICS sets the PCUE_NO_RESUME flag to ignore any resume address that the exit supplies in the following situations:
  - XPCTA is invoked for a C or C++ program that is compiled with the XPLINK option.
  - The task control block (TCB) of the application is no longer available.
  - The transaction abend is an AK*xx* abend (relating to a kill request) other than AKKD or AKKE.
- For an AMODE(64) program, CICS sets the PCUE_NO_RESUME_AMODE64 flag to ignore any resume address that the exit supplies if the 64-bit registers are not available at the time of the abend.

When invoked
After an abend and before the environment is modified.

Exit-specific parameters

UEPPCDS
Address of a storage area that contains program- and

terminal-related information, and that can be mapped using the DSECT DFHPCUE. When XPCTA is invoked, the following DFHPCUE fields are significant:

PCUE_CONTROL_BITS
1-byte flag field. A setting of PCUECBTE indicates that the transaction is linked to a terminal.

Flags PCUE_NO_RESUME and PCUE_NO_RESUME_AMODE64 in PCUE_CONTROL_BITS indicate that a resume address is not supported. When set, any return code of UERCMEA from XPCTA is ignored. CICS sets PCUE_NO_RESUME before invoking XPCTA for C and C++ programs compiled with the XPLINK option, when the TCB of the application is no longer available, and for AK*xx* abends other than AKKD or AKKE. CICS sets PCUE_NO_RESUME_AMODE64 for AMODE(64) programs when a resume address is not supported.

PCUE_TASK_NUMBER
3-character packed decimal field that contains the task number.

PCUE_TRANSACTION_ID
4-character field that contains the transaction ID.

PCUE_TERMINAL_ID
4-character field that contains the terminal ID (if any).

PCUE_PROGRAM_NAME
8-character field that contains the name of the failing program.

PCUE_LOGICAL_LEVEL
Fullword that contains the program logical level.

PCUE_BRANCH_ADDRESS
Fullword. You can use this field to supply a resume address. Set the top bit to specify that the resumed task is to run AMODE(31). Set the bottom bit to specify that the resumed task is to run AMODE(64).

PCUE_BRANCH_EXECKEY
If storage protection is active, you can use this 1-byte field to specify the execution key of the resumed task. The possible values are:

PCUE_BRANCH_USER
User key

PCUE_BRANCH_CICS
CICS key.

If storage protection is active and you do not specify a value, the resumed task executes in user key.

If storage protection is not active, the resumed task executes in CICS key.

UEPTACB
Address of the transaction abend control block (TACB) for the

abend. If the abend occurred because of a program check, the information in the TACB includes:

- The program status word (PSW).
- The registers at the time of the abend.
- The execution key at the time of the abend.
- Details of the subspace and access registers current at the time of the abend.
- The Breaking Event Address Register (BEAR).

You can map the TACB using the DFHTACB TYPE=DSECT macro.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

UERCMEA
A resume address is supplied.

XPI calls
All can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The sample XPCTA global user exit program, DFH$PCTA

The sample program tests whether the abend was caused by the application program trying to overwrite CICS-key storage in the CDSA, ECDSA, ETDSA, or GCDSA while running in user key. If this was the case, the sample changes the execution key to CICS, and retries the failing instruction.

You can use the sample program to identify, without abending, those programs that need to be defined with EXECKEY(CICS), because they intentionally modify a CICS-key DSA. For details of how to do this, see the prolog of DFH$PCTA.

DFH$PCTA can be extended for transaction isolation.

Related concepts:

"Transaction abend sample exit program (DFH$PCTA)" on page 30
CICS includes a sample global user exit program for the XPCTA exit point. The sample program is DFH$PCTA.

Exit XPCABND

Exit XPCABND is invoked after a transaction abend and before a transaction dump call. You can use this exit to suppress the dump.

When invoked
After a transaction abend and before a transaction dump call is made.

Exit-specific parameters

UEPPCDS
Address of a storage area that contains program-related and terminal-related information. The storage area is mapped by the DSECT DFHPCUE.

When XPCABND is invoked, the following DFHPCUE fields are significant:

PCUE_CONTROL_BITS
A 1-byte flag field. A setting of PCUECBTE indicates that the transaction is linked to a terminal.

PCUE_TASK_NUMBER
A 3-character packed decimal field that contains the task number.

PCUE_TRANSACTION_ID
A 4-character field that contains the transaction ID.

PCUE_TERMINAL_ID
A 4-character field that contains the terminal ID (if any).

PCUE_PROGRAM_NAME
An 8-character field that contains the name of the program that is abending.

PCUE_LOGICAL_LEVEL
Fullword that contains the program logical level.

UEPTACB
Address of the transaction abend control block (TACB) for the abend. If the abend occurred because of a program check, the information in the TACB includes:

- The program status word (PSW).
- The registers at the time of the abend.
- Details of the subspace and access registers current at the time of the abend.
- The Breaking Event Address Register (BEAR).

You can map the TACB using the DFHTACB TYPE=DSECT macro.

Return codes

UERCNORM
Continue processing and make the dump call.

UERCBYP
Suppress the dump call.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Resource manager interface program exits (XRMIIN, XRMIOUT)

These exits are invoked when RMI API requests are processed.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XRMIIN

Exit XRMIIN is invoked before a task-related user exit program is invoked when an application program issues an RMI API request.

Exit-specific parameters

UEPTRUEN
Address of the name of the task-related user exit program.

UEPTRUEP
Address of the parameter list to be passed to the task-related user exit program.

UEP_RM_PBTOK
Address of a 4-byte field containing the z/OS Workload Manager (WLM) Performance Block Token. An exit program can use this token to access information (such as the service class token, SERVCLS) in the WLM Performance Block. To do so, it must use the WLM EXTRACT macro, IWMMEXTR, passing the Performance Block Token as the MONTKN input parameter. For more information about the IWMMEXTR macro, see *z/OS MVS Programming: Workload Management Services*.

An exit program must not attempt to modify the Performance Block: if it does so, the results are unpredictable.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

Note: The task-related user exit program's parameter list is mapped by a DFHUEPAR DSECT that shares common field names with the global user exit program's DFHUEPAR parameter list. To include both DSECT definitions in your exit program, you must code:

```
DFHUEXIT TYPE=EP,ID=XRMIIN
DFHUEXIT TYPE,TYPE=RM
```

The statements must be coded in this order.

The two DFHUEPAR parameter lists, the global user exit's and the task-related user exit's, occupy separate areas of storage. The task-related user exit's parameter list is provided for information only; you should not amend it in any way.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

API and SPI commands
All except EXEC CICS SHUTDOWN and EXEC CICS XCTL can be used. However, CALLDLI, EXEC DLI, or EXEC SQL commands must **not** be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XRMIOUT

Exit XRMIOUT is invoked after a task-related user exit program has returned from handling an RMI API request.

Exit-specific parameters

UEPTRUEN
Address of the name of the task-related user exit program.

UEPTRUEP
Address of the parameter list to be passed to the task-related user exit program.

The UEPHMSA parameter in this parameter list contains the register save area (RSA) of the caller. For an EXEC CPSM call, register 1 in this RSA contains the address of the parameter list for the CPSM command and can be used to identify the CPSM command and the specified resource. For details, see CICSPlex SM API command argument list in Reference ->System programming.

UEP_RM_PBTOK
Address of a 4-byte field containing the z/OS Workload Manager (WLM) Performance Block Token. An exit program can use this token to access information (such as the service class token, SERVCLS) in the WLM Performance Block. To do so, it must use the WLM EXTRACT macro, IWMMEXTR, passing the Performance Block Token as the MONTKN input parameter. For more information about the IWMMEXTR macro, see z/OS MVS Programming: Workload Management Services.

An exit program must not attempt to modify the Performance Block: if it does so, the results are unpredictable.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

Note: The task-related user exit program's parameter list is mapped by a DFHUEPAR DSECT that shares common field names with the global user exit program's DFHUEPAR parameter list. To include both DSECT definitions in your exit program, you must code:

```
DFHUEXIT TYPE=EP,ID=XRMIOUT
DFHUEXIT TYPE,TYPE=RM
```

The statements must be coded in this order.

The DFHUEPAR parameter list of the global user exit and the DFHUEPAR parameter list of the task-related user exit each occupy separate areas of storage. The parameter list of the task-related user exit is provided for information only; do not amend it in any way.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

API and SPI commands
All except EXEC CICS SHUTDOWN and EXEC CICS XCTL can be used. However, CALLDLI, EXEC DLI, or EXEC SQL commands must **not** be used.

Note: It is not advisable for your exit program to make calls to other external resource managers that use the RMI, because this causes recursion, and might result in a loop. It is the responsibility of your exit program to avoid entering a loop. You can use the recursion counter field UEPRECUR in the exit program to guard against this possibility.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Resource management installation and discard exit XRSINDI

The XRSINDI global user exit is driven, if it is enabled, immediately after CICS successfully installs or discards a resource definition.

The installation and discard activities that drive the exit are as follows:

- The installation function of the group list on an initial or cold start of CICS
- The **CEDA INSTALL** command
- The CICSPlex SM BAS **INSTALL** command
- All autoinstall operations, as follows:
  - The autoinstall of a terminal, connection, program, map set, partition set, or journal
  - The automatic discard of an unused terminal, controlled by the AILDELAY system initialization parameter and the SIGNOFF attribute on the TYPETERM resource definition.
- The connection to, and disconnection from, an MVS log stream
- A **CEMT DISCARD** and **EXEC CICS DISCARD** command
- An **EXEC CICS CREATE** command
- The front-end programming interface (FEPI) installation and discard operations: the **EXEC CICS FEPI INSTALL** command and **EXEC CICS FEPI DISCARD** command.

The parameter list is designed to pass the names of more than one resource, installed or discarded, in field UEPIDNAM. When designing your global user exit program, do not assume that the number of resource names passed is always one. Analyze the resources in a loop based on the value referenced by UEPIDNUM.

The names of modegroups are prefixed with the corresponding connection name. There is no separator between the two names: the first 4 characters form the connection name, followed by 8 characters for the modegroup. The parts of the concatenated name are fixed length—if connection names are defined with less than 4 characters, they are padded with blanks in the concatenated names. Similarly, the connection names for a front-end programming interface (FEPI) connection is a concatenation of a FEPI node name and a FEPI target name, each of which is 8 characters long (fixed length) with no separator.

The exit is driven once for each individual resource in a group list installed during a CICS initial or cold start. If you are concerned about the performance on an initial or cold start, do not enable the exit until after the group list is installed. To obtain the information about resources installed before enabling the exit, you can write a program to scan the tables of installed resources, by using the **EXEC CICS INQUIRE** *resource_name* browse function.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XRSINDI

The XRSINDI global user exit is called when CICS installs or discards a resource definition.

Exit XRSINDI parameters and return codes

Abends in a program that is enabled at the XRSINDI exit point might cause CICS to shut down, because for some resources the exit is driven during sync point. If the exit returns code UERCPURG during the sync point for these resources, abend code AUEP is produced and CICS shuts down.

The parameters, return codes, and XPI information are as follows:

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Address of the 4-byte terminal ID.

UEPPROG
Address of the 8-byte application program name.

UEPIDREQ
Address of the 1-byte install or discard identifier. The values are as follows:

UEIDINS
This request is for an install action, or, in the case of a log stream, it is a connection to a log stream.

UEIDDIS
This request is for a discard action, or, in the case of a log stream, it is a disconnection from a log stream.

UEPIDTYP
Address of the 1-byte type of resource. The values are as follows:

UEIDATOM
An ATOMSERVICE resource.

UEIDAITM
An autoinstall terminal model.

UEIDBNDL
A BUNDLE resource.

UEIDCONN
A connection.

UEIDDB2C
A DB2CONN resource definition for the connection between CICS and DB2.

UEIDDB2E
A DB2ENTRY resource definition.

UEIDDB2T
A DB2TRAN resource definition.

UEIDDOCT
A DOCTEMPLATE.

UEIDEBAB
An EBA file that is part of a CICS bundle.

UEIDEPAD
An EPADAPTER resource.

UEIDEPAS
An EPADAPTERSET resource.

UEIDEVCS
An event capture resource.

UEIDEVNT
An EVENTBINDING resource.

UEIDFECO
A FEPI connection.

UEIDFENO
A FEPI node.

UEIDFEPO
A FEPI pool.

UEIDFEPS
A FEPI property set.

UEIDFETA
A FEPI target.

UEIDFILE
A file.

UEIDIPCO
An IPCONN resource.

UEIDJNMD
A journal model.

UEIDJNNM
A journal name.

UEIDJSRV
A JVM server resource.

UEIDLBRY
A LIBRARY resource.

UEIDMAP
A mapset.

UEIDMODE
A modegroup.

UEIDMPPP
A Policy resource.

UEIDMQCN
An MQCONN resource definition for the connection between CICS and WebSphere® MQ.

UEIDMQIN
An MQINI resource.

UEIDNQRN
An ENQMODEL.

UEIDOSGB
An OSGi bundle.

UEIDPART
A partner.

UEIDPIPE
A pipeline (PIPELINE).

UEIDPROF
A profile.

UEIDPROG
A program.

UEIDPRTY
A BTS process type.

UEIDPSET
A partition set.

UEIDSESS
A session.

UEIDSTRM
An MVS log stream.

UEIDTCLS
A transaction class.

UEIDTCPS
A TCP/IP service.

UEIDTDQU
A transient data queue.

UEIDTERM
A terminal.

UEIDTRAN
A transaction.

UEIDTSMD
A temporary storage queue model.

UEIDURIM
A URIMAP resource.

UEIDWARB
A WAR file that is part of a CICS bundle.

UEIDWEBS
A web service (WEBSERVICE).

UEIDXMLT
An XMLTRANSFORM resource.

UEPIDLEN
For public resources, this parameter is the address of the length of an individual resource name, as a fullword binary value.

For OSGi bundles, this parameter is the address of the length of the information that uniquely identifies an OSGi bundle in CICS in the **UEPIDNAM** parameter. The maximum length is 526 bytes.

UEPIDNUM
Address of the number of resources that are reported by this call, as a fullword binary value.

UEPIDNAM
Address of a variable-length list containing the names of the individual resources reported by this call.

For OSGi bundles, this parameter contains the information that uniquely identifies an OSGi bundle in CICS. The information is listed in the following order:

1. 8 bytes that contain the JVM server name.
2. A fullword containing the length of the OSGi bundle symbolic name.
3. A fullword containing the length of the OSGi bundle version.
4. A concatenation of the OSGi bundle symbolic name and version as a character string.

UEPIDREC
Address of a 1-byte identifier indicating whether resources are recovered at a warm or emergency restart. The values are as follows:

UEIDKEEP
The resources are recoverable at a warm or emergency restart.

UEIDLOSE
The resources are not recoverable.

Note: The exit is not driven during a CICS restart.

UEPDEFTM
The address of a variable-length list, which corresponds to the list in UEPIDNAM, containing the definition time of the individual resource as an 8-character STCK value.

Note: The parameters UEPDEFTM, UEPCHUSR, UEPCHAGT, UEPCHREL, UEPCHTIM, UEPDEFSRC, UEPINUSR, UEPINTIM, and UEPINAGT are valid for the following resources: ATOMSERVICE, BUNDLE, CONNECTION, DB2CONN, DB2ENTRY, DB2TRAN, DOCTEMPLATE, ENQMODEL, EPADAPTER, EPADAPTERSET, EVENTBINDING, FILE, IPCONN, JOURNALMODEL, JVMSERVER, LIBRARY, MQCONN, MQINI, OSGIBUNDLE, PIPELINE, PROFILE, PROCESSTYPE, PROGRAM, TCPIPSERVICE, TDQUEUE, TRANCLASS, TRANSACTION, TSMODEL, URIMAP, WEBSERVICE, and XMLTRANSFORM. The parameter value is zero for all other resources.

UEPCHUSR
Address of a variable-length list, which corresponds to the list in UEPIDNAM, containing the 8-character user ID that ran the agent that last changed the individual resource.

UEPCHAGT
Address of a variable-length list, which corresponds to the list in UEPIDNAM, of a 2-byte identifier representing the agent that last changed the individual resource. The possible values are as follows:

UEPUNKAGT
The resource was changed by an unknown agent.

UEPCSDAPI
The resource was changed using the CSD API or CEDA.

UEPCSDBAT
The resource was changed using the CSD batch program, DFHCSDUP.

UEPDRPAPI
The resource was changed using the CICSPlex SM BAS API.

UEPAUTOIN
The resource was changed using autoinstall.

UEPSYSTEM
The resource was changed by the running CICS region.

UEPDYNAMC
The resource was changed dynamically.

UEPTABLE
The resource was changed using a table.

UEPCHREL
Address of a variable-length list, which corresponds to the list in UEPIDNAM, containing the 4-character CICS release level that was running when the individual resource was last changed.

UEPCHTIM
Address of a variable-length list, which corresponds to the list in UEPIDNAM, containing the CSD record time stamp change for the individual resource as an 8-character STCK value.

UEPDEFSRC
Address of a variable-length list, which corresponds to the list in UEPIDNAM, containing the 8-character CSD group name or source corresponding to the individual resource.

UEPINUSR
Address of a variable-length list, which corresponds to the list in UEPIDNAM, containing the 8-character user ID that installed the individual resource.

UEPINTIM
Address of a variable-length list, which corresponds to the list in UEPIDNAM, containing the time that the domain was called for the installation of the individual resource as an 8-character STCK value.

UEPINAGT
Address of a variable-length list, which corresponds to the list in UEPIDNAM, of a 2-byte identifier representing the agent that installed the individual resource. The possible values are as follows:

UEPCSDAPI
The resource was installed using the CSD API or CEDA.

UEPCRESPI
The resource was installed using the EXEC CICS CREATE SPI commands.

UEPGRPLST
The resource was installed at startup using GRPLIST install.

UEPAUTOIN
The resource was autoinstalled.

UEPSYSTEM
The resource was installed by the running CICS system.

UEPDYNAMC
The resource was installed dynamically.

UEPBUNDLE
The resource was installed by a bundle deployment.

UEPTABLE
The resource was installed using a table.

UEPAPPTK
Address of a variable-length list, containing an 8-character token representing the application instance to which this resource belongs. For public resources, this address is zero.

UEPAPCTXT
For private resources for applications that are deployed on platforms, this parameter contains the address of a variable-length list, which corresponds to the list in UEPIDNAM, containing the application context information for the resource. The information is listed in the following order:

1. The private resource name, padded with spaces to 8 characters.
2. The platform name, padded with spaces to 64 characters.
3. The application name, padded with spaces to 64 characters.
4. The major version number for the application, which is a fullword binary value.
5. The minor version number for the application, which is a fullword binary value.
6. The micro version number for the application, which is a fullword binary value.

CICS supplies a DSECT named **DFHUEACD** which maps this information. For more information about **DFHUEACD**, see UEACD - User exit application context in Data Areas.

Return codes

UERCNORM
Continue processing. This code is the default.

UERCPURG
Task purged during XPI call.

XPI calls
You can use all XPI calls.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell

you whether the call was successful.

Signon and signoff exits XSNON, XSNOFF, and XSNEX

Exit XSNON is invoked after a terminal user signs on, and exit XSNOFF is invoked after a terminal user signs off (whether the signon or sign-off is successful or not). XSNON and XSNOFF do not make any security decisions; they are merely a means of tracking users logging on and off a CICS system.

The activities which drive the exits are:

- Invocation of an EXEC CICS SIGNON command for a terminal (when, for example, the terminal user enters the CICS-supplied CESN, or an equivalent, user-written, signon transaction)
- Invocation of an EXEC CICS SIGNON command for a surrogate terminal (that is, a terminal attached by the CRTE routing transaction, or by dynamic transaction routing)
- Invocation of an EXEC CICS SIGNOFF command for a terminal
- When a 'CANCEL' command is entered to terminate a CRTE routing session
- A timeout sign-off.

XSNEX is a special-purpose global user point, which is intended to be used only with the IBM-supplied global user exit program, DFH$SNEX.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XSNON

When invoked

When a user signs on.

Exit-specific parameters

UEPUSRID
Address of the terminal userid.

UEPUSRLN
Address of the terminal userid length.

UEPGRPID
Address of the group ID. If the signon was successful, the group ID is that which the user is associated with in this signon session. If the signon was unsuccessful, it is that specified by the user when he or she tried to sign on.

UEPGRPLN
Address of the group ID length.

UEPNETN
Address of the terminal's netname.

UEPTRMID
Address of the terminal id.

UEPTCTUA
Address of the TCT user area.

UEPTCTUL
Address of the TCT user area length.

UEPTRMTY
Address of the terminal-type byte.

UEPSNFLG
Address of a 2-byte field containing flags:

Table 1. Flags set in the UEPSNFLG field of XSNON

Flag	Equivalent	Meaning
UEPSNOK	0	Signon was successful.
UEPSNFL	1	Signon failed.
UEPSNPSS	2	The persistent sessions signon succeeded.
UEPSNPSF	3	The persistent sessions signon failed.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XSNOFF

When invoked
When a user signs off.

Exit-specific parameters

UEPUSRID
Address of the terminal userid.

UEPUSRLN
Address of the terminal userid length.

UEPGRPID
Address of the group ID.

UEPGRPLN
Address of the group ID length.

UEPNETN
Address of the terminal's netname.

UEPTRMID
Address of the terminal id.

UEPTCTUA
Address of the TCT user area.

UEPTCTUL
Address of the TCT user area length.

UEPTRMTY
Address of the terminal-type byte.

UEPSNFLG
Address of a 2-byte field containing flags:

UEPSNOK
Sign-off was successful

UEPSNFL
Sign-off failed

UEPSNNML
Normal sign-off

UEPSNTIM
Timeout sign-off.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XSNEX

The purpose of XSNEX, in conjunction with its supporting sample programs, is to provide a short-term aid for upgrading. It is designed to give you time to modify those application programs that have a dependency on the way CICS handles **EXEC CICS SIGNON** and SIGNOFF before CICS TS 2.1, to enable them to work with the current behavior.

Note: XSNEX is for upgrade purposes only. Remove all application dependency on the old sign-on and sign-off behavior.

There are no exit-specific parameters for this global user exit, which is invoked whenever an application program issues an **EXEC CICS SIGNON** or an **EXEC CICS SIGNOFF** command. You are not intended to write your own global user exit

program for this exit point. IBM provides DFH$SNEX, the sole purpose of which is to make CICS handle **EXEC CICS SIGNON** and SIGNOFF commands in the same way as in CICS TS 1.3 and earlier.

The supplied programs are:

DFH$SNEX
This user exit program is supplied in SDFHSAMP. The only function the program performs is to set return code UERCPREV, which causes the security domain to restore CICS behavior as in CICS TS 1.3 and earlier. You can enable this user exit program using DFH$SNPI.

DFH$SNPI
This post-initialization program is supplied in SDFHSAMP. It issues an EXEC CICS ENABLE PROGRAM('DFH$SNEX') EXIT('XSNEX') command to enable the IBM-supplied user exit program, DFH$SNEX, in the final stages of CICS initialization.

To use this program, add an entry to the first section of your PLTPI table (that is, before the DFHDELIM statement). For example:

```
DFHPLT TYPE=INITIAL,SUFFIX=SN
DFHPLT TYPE=ENTRY,PROGRAM=DFH$SNPI
DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
DFHPLT TYPE=FINAL
END
```

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Statistics domain exit XSTOUT

On invocation, XSTOUT is passed the address of a buffer containing one or more statistics records. The buffer can contain records for various resource types; for example, connections and modenames. The buffer can also contain both specific and global information; for example, loader statistics for individual programs, and loader statistics for all programs.

Your exit program can identify the types of records in the buffer by their STID values. (STID values are described in "CICS statistics data section" on page 655.)

You can use XSTOUT to prevent the contents of the statistics data buffer being written to SMF. Note that you cannot use it to selectively suppress individual records within the buffer. Your exit program should not modify the values of any of the exit-specific parameters.

Some statistics records might be produced during very early during CICS initialization which will not be passed to XSTOUT. The earliest that a global user exit can be enabled is during PLT processing. Before this no exits can be invoked.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XSTOUT

When invoked

Before a statistics record is written to SMF.

Exit-specific parameters

Fields UEPPROG, UEPTERM, UEPTRANID, and UEPUSER have meaning only for requested statistics (when using the CICS Explorer® **Regions** operations view, the **CEMT PERFORM STATISTICS RECORD** command, or the **EXEC CICS PERFORM STATISTICS RECORD** command).

UEPPROG

Address of the 8-byte application program name.

UEPSCLD

Address of an 8-byte character field containing the collection date (MMDDYYYY).

UEPSDATE

Address of a 6-byte character field containing the collection date (MMDDYY).

UEPSIVAL

Address of a 6-byte character field containing the interval time (HHMMSS). This field has meaning only for interval statistics.

UEPSIVN

Address of the 4-byte interval number. This field has meaning only for interval statistics.

UEPSRLEN

Address of the 4-byte hexadecimal length of the statistics record.

UEPSTATS

Address of a buffer containing one or more statistics records. For unsolicited statistics, the buffer always contains one record; for other types of statistics, it might contain several records. The length of the buffer is addressed by the UEPSRLEN parameter.

UEPSTIME

Address of a 6-byte character field containing the collection time (HHMMSS).

UEPSTYPE

Address of the 3-byte character field statistics type. The values of the types are as follows:

INT Interval statistics.

EOD End-of-day statistics.

REQ Requested statistics.

RRT Requested reset statistics.

USS Unsolicited statistics.

UEPTERM

Address of the 4-byte terminal ID.

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

Return codes

UERCBYP
Suppress output of statistics data buffer to SMF.

UERCNORM
Continue processing.

XPI calls
WAIT_MVS can be used. Note, however, that the wait cannot be purged by using CEMT or SPI commands. Do not use any other calls.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

System recovery program exit XSRAB

Exit XSRAB is invoked when the system recovery program (DFHSRP) finds a match in the system recovery table (SRT) for an operating system abend code.

When invoked
Exit XSRAB is invoked when the system recovery program (DFHSRP) finds a match in the SRT for an operating system abend code. For information about defining entries in the SRT, see System recovery table (SRT).

The SRT table is processed and the exit is driven, only when an MVS abend occurs under a CICS essential TCB; that is, one of QR, RO, CO, SZ, RP, or FO. For nonessential TCB types, such as L8, SL, SO, or S8, the exit is not driven.

Exit-specific parameters

UEPERROR
Address of the error data structure, SRP_ERROR_DATA, which contains the following fields:

SRP_ERROR_TYPE
The 4-character error type, which is always ASRB.

SRP_SYS_ABCODE
2 bytes that contain the system abend code XXX in binary format (for example, D37).

SRP_USER_ABCODE
2 bytes that contain the user abend code NNNN in binary format (for example, 0999).

SRP_ERROR_TRANID
4-character field that contains the ID of the transaction that abended.

SRP_ERROR_STACK_NAME
8-character field that contains the name of the current kernel stack entry for the transaction at the time of the abend.

SRP_ERROR_PPT_NAME
8-character field that contains the name of the current program for the transaction. This field contains a value only if flag SRP_PPT_ENTRY is set.

SRP_ERROR_OFFSET
Fullword that contains the offset into the program that abended, as follows:

- If flag SRP_PPT_ENTRY is set, gives the offset in SRP_ERROR_PPT_NAME
- Otherwise, gives the offset in SRP_ERROR_STACK_NAME.

This field contains a value only if flag SRP_VALID_OFFSET is set.

SRP_ERROR_FLAGS
1 byte that contains flags:

SRP_CICS_CODE
The abend occurred while running CICS code.

SRP_USER_CODE
The abend occurred while running user application code.

SRP_PPT_ENTRY
The abend occurred while running SRP_ERROR_PPT_NAME. If this flag is not set, the abend occurred while running SRP_ERROR_STACK_NAME.

SRP_VALID_OFFSET
A meaningful offset could be determined.

SRP_VALID_REASON
MVS has supplied a reason code for the abend.

SRP_NOT_CICS_RB
CICS RB was not in control at the time of the abend (that is, the abend occurred in a system service invoked by CICS).

SRP_CICS_ERROR_REASON
4-character field that contains the MVS abend reason code. It contains a value only if flag SRP_VALID_REASON is set.

SRP_CICS_ERROR_DATA
An area that describes the last thing that CICS did, before the abend. It contains the following:

SRP_CICS_EC_PSW
8-character field that contains the extended control (EC) mode program status word (PSW)

SRP_CICS_PSW16
16-character field that contains the 128-bit PSW

SRP_CICS_EC_INT
8-character field that contains the interrupt code and ILC

SRP_CICS_REGST
64-character field that contains the contents of the general-purpose (GP) registers

SRP_CICS_EXEC_KEY
1 byte that contains the PSW key, in the form X'0n'.

SRP_SYSTEM_ERROR_DATA
An area that describes the last thing the system did, before the abend. It contains the following:

SRP_SYSTEM_EC_PSW
8-character field that contains the EC mode PSW

SRP_SYSTEM_PSW16
16-character field that contains the 128-bit PSW

SRP_SYSTEM_EC_INT
8-character field that contains the interrupt code and ILC

SRP_SYSTEM_REGST
64-character field that contains the contents of the GP registers

SRP_SYSTEM_EXEC_KEY
1 byte that contains the PSW key, in the form X'0n'.

SRP_ERROR_FP_REGS
An area that describes the contents of the floating point registers at the time of the abend. It contains:

SRP_FP_REG_0
FP register 0

SRP_FP_REG_2
FP register 2

SRP_FP_REG_4
FP register 4

SRP_FP_REG_6
FP register 6

SRP_ADDITIONAL_REG_INFO
An area that contains additional register information.

SRP_ADDITIONAL_REGS_FLAG
1 byte that contains flags:

SRP_CICS_GPR64_AVAIL
The 64-bit CICS GP registers are available.

SRP_SYSTEM_GPR64_AVAIL
The 64-bit system GP registers are available.

SRP_ADDITIONAL_FPR_AVAIL
Additional FP registers are available.

SRP_CICS_GP64_REGS
128-byte area that contains the CICS 64-bit GP registers at the time of the abend.

SRP_SYSTEM_GP64_REGS
128-byte area that contains the system 64-bit GP registers at the time of the abend.

SRP_ADDITIONAL_FPR_REGS
132-byte area that contains additional FP registers at the time of the abend.

SRP_FP_REGS
128-byte area that contains all the FP registers at the time of the abend.

SRP_FPC_REG
4-byte field that contains the FPC register at the time of the abend.

If flag SRP_NOT_CICS_RB is set, SRP_CICS_ERROR_DATA describes the last thing that CICS did before the abend and SRP_SYSTEM_ERROR_DATA describes the last thing that the system service (for example, z/OS Communications Server, VSAM, or MVS) did.

You can map the SRP_ERROR_DATA that is passed to the XSRAB exit by using the DFHSRED TYPE=DSECT macro. The format of SRP_ERROR_DATA is shown in SRED - System recovery error data in Reference -> Diagnostics.

Return codes

UERCNOCA
Abnormally terminate the task with abend code ASRB. Do not cancel any program-level abend exits that are associated with this task.

UERCCANC
Abnormally terminate the task with abend code ASRB. Cancel any program-level abend exits that are associated with this task.

UERCCICS
Abnormally terminate CICS.

XPI calls

Because CICS invokes the exit XSRAB in an error environment, you can use only a subset of the XPI calls.

Only TRACE_PUT is available for general use.

You can use WAIT_MVS, but only after the exit program determines (from the SRP_CICS_CODE and SRP_USER_CODE fields) that the abend occurred in user application code, and not in CICS code.

Important:

- Take care when coding a program to run at the XSRAB exit point. If your exit program causes the system recovery program to be reentered (for example, if a program check occurs), CICS terminates abnormally with a DFHSR06*xx* message.
- The default return code is UERCNOCA, which ensures that the task abends if the exit is in error.
- There is no UERCNORM return code at this exit point, because the exit is invoked after a failure.
- The exit should not set the return code UERCPURG.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

System termination program exit XSTERM

The XSTERM exit could be used to output final statistics to your statistics SMF data sets, and to close them. Note that CICS VSAM and BDAM data sets have already been closed by CICS file control before the exit is invoked.

When invoked

During the second quiesce stage of a normal system shutdown, immediately before the transient data and temporary storage buffers are cleared. The exit is not invoked during an IMMEDIATE shutdown.

Exit-specific parameters

None.

Return codes

UERCNORM

Continue processing.

XPI calls

All other XPI calls except WRITE_JOURNAL_DATA can be used. However, their use is not recommended, because they could cause the task to lose control, thus allowing another task to write more monitoring data.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Temporary storage domain exits (XTSQRIN, XTSQROUT, XTSPTIN, XTSPTOUT)

These exits are invoked when temporary storage has to be controlled or monitored.

You can change the temporary storage domain exits XTSQRIN, XTSQROUT, XTSPTIN, and XTSPTOUT to perform the following tasks:

- Specify, for a request that creates a queue, whether the queue is to be held in main or auxiliary storage, and its recoverability
- Monitor the use of temporary storage
- Control security for temporary storage queues

The UEPTERM parameter is a zero value for temporary storage requests that have been function shipped over an IPIC connection. To use IPIC connections for temporary storage requests, ensure that XTSQRIN, XTSQROUT, XTSPTIN, and XTSPTOUT check that the UEPTERM parameter is a non-zero value before trying to use it as an address.

XTSQRIN, XTSQROUT, XTSPTIN, and XTSPTOUT must be coded to threadsafe standards and declared threadsafe to get the benefits of being threadsafe using an IPIC connection.

The temporary storage domain has two main gates, TSQR and TSPT, that support the following functions:

TSQR Write, Rewrite, Read_into, Read_set, Read_next_into, Read_next_set, Delete

TSPT Put, Put_replace, Get, Get_set, Get_release, Get_release_set, Release

The TSQR functions correspond to those available through the EXEC CICS interface (or through DFHTS TYPE=PUTQ, GETQ, or PURGE). The TSPT functions are used by the interval control program in support of START and RETRIEVE functions (or DFHTS TYPE=PUT, GET, or RELEASE).

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XTSQRIN

Exit XTSQRIN is invoked before execution of a user temporary storage interface request for a user TS queue (for example, a WRITEQ TS, or READQ TS request).

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Zero, or the address of the 4-byte terminal ID. If no address is returned, this could mean that this request has been function shipped over an IPIC connection.

UEPPROG
Address of the 8-byte application program name.

UEP_TS_FUNCTION
Address of a byte containing the function:

- UEP_TS_FUN_WRITE
- UEP_TS_FUN_REWRITE
- UEP_TS_FUN_READ_INTO
- UEP_TS_FUN_READ_SET
- UEP_TS_FUN_READ_NEXT_INTO
- UEP_TS_FUN_READ_NEXT_SET
- UEP_TS_FUN_DELETE

UEP_TS_QUEUE_NAME
Address of a 16-byte field containing the queue name.

UEP_TS_DATA_P
Address of a fullword containing the address of the data. (Write and rewrite requests).

UEP_TS_DATA_L
Address of a fullword containing the length of the data. (Write and rewrite requests).

UEP_TS_ITEM_NUMBER
Address of a fullword containing the item number. (Rewrite, read_into and read_set requests).

UEP_TS_STORAGE_TYPE
Address of a byte containing the storage type. (Write requests).

On input to the exit, the parameter will be set to either UEP_TS_STORAGE_TYPE_MAIN or UEP_TS_STORAGE_TYPE_AUX_TST. This parameter may be modified by the exit to any of the following values.

Note that if CICS has been initialized with TS main-only support, setting this parameter has no effect.

UEP_TS_STORAGE_TYPE_MAIN
Main storage.

UEP_TS_STORAGE_TYPE_AUX_TST
Auxiliary storage (recoverability determined by the resource definition).

UEP_TS_STORAGE_TYPE_AUX_RECOV_YES
Auxiliary storage (recoverable).

UEP_TS_STORAGE_TYPE_AUX_RECOV_NO
Auxiliary storage (non-recoverable).

Return codes

UERCNORM
Normal.

UERCPURG
Purged.

XPI calls
All can be used.

API and SPI calls
None can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XTSQROUT

Exit XTSQROUT is invoked after execution of a user temporary storage interface request for a user TS queue (for example, a WRITEQ TS, or READQ TS request).

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Zero, or the address of the 4-byte terminal ID. If no address is returned, this could mean that this request has been function shipped over an IPIC connection.

UEPPROG
Address of the 8-byte application program name.

UEP_TS_FUNCTION
Address of a byte containing the function:

- UEP_TS_FUN_WRITE
- UEP_TS_FUN_REWRITE
- UEP_TS_FUN_READ_INTO
- UEP_TS_FUN_READ_SET
- UEP_TS_FUN_READ_NEXT_INTO
- UEP_TS_FUN_READ_NEXT_SET
- UEP_TS_FUN_DELETE

UEP_TS_QUEUE_NAME
Address of a 16-byte field containing the queue name.

UEP_TS_DATA_P
Address of a fullword containing the address of the data. (All requests except delete).

UEP_TS_DATA_L
Address of a fullword containing the length of the data. (All requests except delete).

UEP_TS_ITEM_NUMBER
Address of a fullword containing the item number. (Rewrite, read_into and read_set requests).

UEP_TS_TOTAL_ITEMS
Address of a fullword containing the total number of items in the queue. (All requests except delete).

UEP_TS_RESPONSE
Address of a byte containing the response after a request has been completed.

- UEP_TS_RESPONSE_OK
- UEP_TS_RESPONSE_PURGED
- UEP_TS_RESPONSE_EXCEPTION
- UEP_TS_RESPONSE_DISASTER
- UEP_TS_RESPONSE_INVALID

Return codes

UERCNORM
Normal response.

UERCPURG
A purged response was received from an XPI request.

XPI calls
All can be used.

API and SPI calls
None can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XTSPTIN

Exit XTSPTIN is invoked before execution of a temporary storage interface request for a CICS internal queue (for example, for interval control or BMS queues).

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Zero, or the address of the 4-byte terminal ID. If no address is returned, this could mean that this request has been function shipped over an IPIC connection.

UEPPROG
Address of the 8-byte application program name.

UEP_TS_FUNCTION
Address of a byte containing the function:

- UEP_TS_FUN_PUT
- UEP_TS_FUN_PUT_REPLACE
- UEP_TS_FUN_GET
- UEP_TS_FUN_GET_SET
- UEP_TS_FUN_GET_RELEASE
- UEP_TS_FUN_GET_RELEASE_SET
- UEP_TS_FUN_RELEASE

UEP_TS_QUEUE_NAME
Address of a 16-byte field containing the queue name.

UEP_TS_DATA_P
Address of a fullword containing the address of the data. (Put and put_replace).

UEP_TS_DATA_L
Address of a fullword containing the length of the data. (Put and put_replace).

UEP_TS_STORAGE_TYPE
Address of a byte containing the storage type. (Put requests).

On input to the exit, the parameter will be set to either UEP_TS_STORAGE_TYPE_MAIN or UEP_TS_STORAGE_TYPE_AUX_TST. This parameter may be modified by the exit to any of the following values.

Note that if CICS has been initialized with TS main-only support, setting this parameter has no effect.

UEP_TS_STORAGE_TYPE_MAIN
Main storage.

UEP_TS_STORAGE_TYPE_AUX_TST
Auxiliary storage (recoverability determined by the resource definition).

UEP_TS_STORAGE_TYPE_AUX_RECOV_YES
Auxiliary storage (recoverable).

UEP_TS_STORAGE_TYPE_AUX_RECOV_NO
Auxiliary storage (non-recoverable).

Return codes

UERCNORM
Normal.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

API and SPI calls
None can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XTSPTOUT

Exit XTSPTOUT is invoked after execution of a temporary storage interface request for a CICS internal queue (for example, for interval control or BMS queues). After execution of a TSPT request. No parameters may be modified.

Exit-specific parameters

UEPTRANID
Address of the 4-byte transaction ID.

UEPUSER
Address of the 8-byte user ID.

UEPTERM
Zero, or the address of the 4-byte terminal ID. If no address is returned, this could mean that this request has been function shipped over an IPIC connection.

UEPPROG
Address of the 8-byte application program name.

UEP_TS_FUNCTION
Address of a byte containing the function:

- UEP_TS_FUNCTION_PUT
- UEP_TS_FUN_PUT_REPLACE
- UEP_TS_FUN_GET
- UEP_TS_FUN_GET_SET
- UEP_TS_FUN_GET_RELEASE
- UEP_TS_FUN_GET_RELEASE_SET
- UEP_TS_FUN_RELEASE

UEP_TS_QUEUE_NAME
Address of a 16-byte field containing the queue name.

UEP_TS_DATA_P
Address of a fullword containing the address of the data. (All requests except release).

UEP_TS_DATA_L
Address of a fullword containing the length of the data. (All requests except release).

UEP_TS_RESPONSE
Address of a byte containing the response after a request has been completed.

- UEP_TS_RESPONSE_OK
- UEP_TS_RESPONSE_PURGED
- UEP_TS_RESPONSE_EXCEPTION

- UEP_TS_RESPONSE_DISASTER
- UEP_TS_RESPONSE_INVALID

Return codes

UERCNORM
Normal response.

UERCPURG
A purged response was received from an XPI request.

XPI calls
All can be used.

API and SPI calls
None can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Temporary storage EXEC interface program exits XTSEREQ and XTSEREQC

The XTSEREQ exit allows you to intercept temporary storage API requests before any action has been taken on the request. The XTSEREQC exit allows you to intercept the response after a temporary storage API request has completed.

The API requests affected are:

- **EXEC CICS WRITEQ TS**
- **EXEC CICS READQ TS**
- **EXEC CICS DELETEQ TS.**

Using XTSEREQ, you can:

- Analyze the API parameter list (function, keywords, argument values, and responses)
- Modify any input parameter value before execution of a request
- Prevent execution of a request.

Using XTSEREQC, you can:

- Analyze the API parameter list
- Modify any output parameter value after request completion.

You can also:

- Pass data between your XTSEREQ and XTSEREQC exit programs when they are invoked for the same request
- Pass data between your temporary storage exit programs when they are invoked within the same task.

It is possible that programs invoked from the exits in the temporary storage domain (XTSQRIN, XTSQROUT, XTSPTIN, and XTSPTOUT) could modify situations set up by XTSEREQ; therefore you must consider the order in which the exits are invoked.

If all the temporary storage exits are enabled, the order of invocation is as follows:

1. XTSEREQ
2. XTSQRIN
3. XTSQROUT
4. XTSEREQC

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XTSEREQ

When invoked

Before CICS processes a temporary storage API request.

Exit-specific parameters

UEPCLPS

Address of a copy of the command parameter list. See "The command-level parameter structure" on page 280.

UEPTQTOK

Address of a 4-byte area which can be used to pass information between XTSEREQ and XTSEREQC for a single temporary storage request.

UEPRCODE

Address of a 6-byte hexadecimal copy of the EIB return code EIBRCODE. For details of EIB return codes, see EIB fields, in the *CICS Application Programming Reference*.

UEPRESP

Address of a 4-byte binary copy of the EIB response code EIBRESP.

UEPRESP2

Address of a 4-byte binary copy of the EIB response code EIBRESP2.

UEPTSTOK

Address of a 4-byte token which can be used to pass information between successive temporary storage requests within the same task (for example, between successive invocations of the XTSEREQ exit).

UEPRECUR

Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPRSRCE

Address of an 8-character copy of the EIB resource value, EIBRSRCE.

Return codes

UERCBYP
Bypass this request.

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

API and SPI commands
All can be used, except for:
EXEC CICS SHUTDOWN
EXEC CICS XCTL

Note: Take care when issuing recursive commands. For example, you must avoid entering a loop when issuing a temporary storage request from the XTSEREQ exit. Use of the recursion counter UEPRECUR is recommended.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XTSEREQC

Exit XTSEREQC is invoked after a temporary storage API request has completed, before return from the temporary storage EXEC interface program.

Exit-specific parameters

UEPCLPS
Address of a copy of the command parameter list. See "The command-level parameter structure" on page 280.

UEPTQTOK
Address of a 4-byte area which can be used to pass information between XTSEREQ and XTSEREQC for a single temporary storage request.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code EIBRCODE. For details of EIB return codes, see EIB fields in Reference -> Application development.

UEPRESP
Address of a 4-byte binary copy of the EIB response code EIBRESP.

UEPRESP2
Address of a 4-byte binary copy of the EIB response code EIBRESP2.

UEPTSTOK
Address of a 4-byte token which can be used to pass information

between successive temporary storage requests within the same task (for example, between successive invocations of the XTSEREQC exit).

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

UEP_TS_REMOTE_SYSTEM
If the request is to be sent to a remote region, is the address of an area containing the 4-byte name of the remote region. The remote region might have been specified by, for example, the `SYSID` option of the command, function shipping, or workload management.

If the request is to be executed on the local region, this parameter is the address of a 4-byte area containing blanks.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

API and SPI commands
All can be used, except for:

```
EXEC CICS SHUTDOWN
EXEC CICS XCTL
```

You can update the copies of EIBRSRCE, EIBRCODE, EIBRESP, and EIBRESP2 that you are given in the parameter list. If you update the values, temporary storage copies the new values into the application program's EIB after the completion of XTSEREQC or if you specify a return code of UERCBYP in XTSEREQ.

You must set valid temporary storage responses. You must set all three of EIBRCODE, EIBRESP, and EIBRESP2 to a consistent set of values, such as would be set by temporary storage to describe a valid completion. CICS does not check the consistency of EIBRCODE, EIBRESP, and EIBRESP2. If EIBRCODE is set to a non-zero value and EIBRESP is set to zero, CICS will override EIBRESP with a non-zero value. To help you set values for EIBRCODE, EIBRESP, and EIBRESP2, the values used by temporary storage are specified in DSECT DFHTSUED.

Note: Take care when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when issuing a temporary storage request from the XTSEREQC exit. Use of the recursion counter UEPRECUR is recommended.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The command-level parameter structure

The command-level parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of a bit string that describes the type of request and identifies each keyword specified with the request. The remaining addresses point to pieces of data associated with the request.

End of parameter list indicator

You can examine the EID to determine the type of request and the keywords specified. You can examine the other parameters in the list to determine the values of the keywords. You can also modify values of keywords specified on the request.

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

The UEPCLPS exit-specific parameter:

The UEPCLPS exit-specific parameter is included in both exit XTSEREQ and exit XTSEREQC. It is the address of the command-level parameter structure.

The command-level parameter structure contains 8 addresses, TS_ADDR0 through TS_ADDR7. It is defined in the DSECT TS_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHTSUED.

The command-level parameter list is made up as follows.

Note: The relationship between arguments, keywords, data types, and input/output types is summarized for the temporary storage commands in the following tables:

Table 10. The relationship between arguments, keywords, data types, and input/output types for the temporary storage commands

Command	See
WRITEQ TS	Table 11 on page 284
READQ TS	Table 12 on page 284
DELETEQ TS	Table 13 on page 284

TS_ADDR0
is the address of a 9-byte area called the EID, which is made up as follows:

- **TS_GROUP**
- **TS_FUNCT**
- **TS_BITS1**
- **TS_BITS2**
- **TS_EIDOPT5**

- **TS_EIDOPT6**
- **TS_EIDOPT7**
- **TS_EIDOPT8**

TS_GROUP
Always X'0A', indicating that this is a temporary storage request.

TS_FUNCT
One byte that defines the type of request:

X'02' WRITEQ

X'04' READQ

X'06' DELETEQ

TS_BITS1
Existence bits that define which arguments were specified. To obtain the argument associated with a keyword, you need to use the appropriate address from the command-level parameter structure. Before using this address, you must check the associated existence bit. If the existence bit is set off, the argument was not specified in the request and the address should not be used.

X'80' Set if the request contains an argument for the QUEUE or QNAME keyword. If set, **TS_ADDR1** is meaningful.

X'40' Set if the request contains an argument for any of the FROM, INTO, or SET keywords. If set, **TS_ADDR2** is meaningful.

X'20' Set if the request contains an argument for the LENGTH keyword. If set, **TS_ADDR3** is meaningful.

X'10' Set if the request contains an argument for the NUMITEMS keyword. If set, **TS_ADDR4** is meaningful.

X'08' Set if the request contains an argument for the NUMITEMS or ITEM keyword. If set, **TS_ADDR5** is meaningful.

X'02' Set if the request contains an argument for the SYSID keyword. If set, **TS_ADDR7** is meaningful.

TS_BITS2
Two bytes not used by temporary storage.

TS_EIDOPT5
Indicates whether certain keywords were specified on the request.

X'80' QNAME was specified (otherwise QUEUE). You can modify this bit in your user exit if you want.

TS_EIDOPT6
One byte not used by temporary storage.

TS_EIDOPT7
Indicates whether certain functions and/or keywords were specified on the request.

X'10' WRITEQ NOSUSPEND specified.

X'80' WRITEQ MAIN or READQ ITEM specified.

X'04' WRITEQ REWRITE or READQ NUMITEMS specified.

TS_EIDOPT8
Indicates whether certain keywords were specified on the request.

X'80' ITEM was specified (otherwise NUMITEMS).

TS_ADDR1
is the address of area containing 8-byte name from QUEUE. or 16-byte name from QNAME. To determine which of these is applied, see the TS_BITS2 field.

TS_ADDR2
is the address of one of the following:

- A 4-byte address from SET (if the request is READQ and **TS_EIDOPT5** indicates that this is SET).
- Data from INTO (if the request is READQ and **TS_EIDOPT5** indicates that this is not SET).
- Data from FROM (if the request is WRITEQ).

TS_ADDR3
is the address of the halfword value of LENGTH (if the request is READQ or WRITEQ).

Warning: For requests that specify INTO, do not change the value of LENGTH to a value greater than that specified by the application. To do so causes a storage overlay in the application.

TS_ADDR4
is the address of the halfword value of NUMITEMS (if the request is READQ).

TS_ADDR5
is the address of one of the following:

- The halfword value of NUMITEMS (if the request is WRITEQ)
- The halfword value of ITEM (if the request is READQ or WRITEQ).

TS_ADDR6
is the address of a value intended for CICS internal use only. It must not be used.

TS_ADDR7
is the address of an area containing the value of SYSID.

Modifying fields in the command-level parameter structure:

Some fields that are passed to temporary storage are used as input to the request, some are used as output fields, and some are used for both input and output. The method your user exit program uses to modify a field depends on the usage of the field.

The following are always input fields:

- QUEUE|QNAME
- FROM
- SYSID

The following are always output fields:

- INTO
- NUMITEMS
- SET

LENGTH is an input field on a WRITEQ request, and an output field on a READQ request that specifies SET. It is both an input and an output field on a READQ request that specifies INTO.

ITEM is an input field on a READQ request, and on a WRITEQ request that specifies REWRITE. It is both an input and an output field on a WRITEQ request that does not specify REWRITE.

Modifying input fields:

The correct method of modifying an input field is to create a new copy of it, and to change the address in the command-level parameter list to point to your new data.

Do not modify an input field by altering the data that is pointed to by the command-level parameter list. To do so would corrupt storage belonging to the application program and would cause a failure when the program attempted to reuse the field.

Modifying output fields: The technique described in "Modifying input fields" is not suitable for modifying output fields. (The results would be returned to the new area instead of the application's area, and would be invisible to the application.)

An output field is modified by altering the data that is pointed to by the command-level parameter list. In the case of an output field, you can modify the application's data in place, because the application is expecting the field to be modified anyway.

Modifying fields used for both input and output: An example of a field that is used for both input and output is LENGTH on a READQ request that specifies INTO. You can treat such fields in the same way as output fields, and they are considered to be the same.

Modifying the EID:

It is not possible to modify the EID to make major changes to requests. It is not possible, for example, to change a READQ request to a WRITEQ request. However, you can make minor changes to requests.

The list that follows shows the bits in the EID that can be modified. Any attempt to modify any other part of the EID is ignored.

TS_BITS1

X'02' The existence bit for SYSID.

TS_EIDOPT7

A user exit program at XTSEREQ can set the following on or off for all WRITEQ TS commands:

X'10' The existence bit for NOSUSPEND.

X'08' The existence bit for MAIN.

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the temporary storage request only.

Note: Your user exit program is prevented from making major changes to the EID. However, you must take great care when making the minor modifications that **are** permitted.

Use of the task token UEPTSTOK:

The task token UEPTSTOK provides the address of a 4-byte area that you can use to pass information between successive temporary storage requests in the same task.

For example, you can use UEPTSTOK to pass information between successive invocations of the XTSEREQ exit. By contrast, UEPTQTOK is usable only for the duration of a single temporary storage request, because its contents may be destroyed at the end of the request.

Table 11. WRITEQ TS: User arguments and associated keywords, data types, and input/output types

Argument	Keyword	Data type	Input/output type
Arg1	QUEUE	CHAR(8)	input
Arg1	QNAME	CHAR(16)	input
Arg2	FROM	DATA-AREA	input
Arg3	LENGTH	BIN(15)	input
Arg4	*	*	*
Arg5	ITEM	BIN(15)	input/output
Arg5	NUMITEMS	BIN(15)	output
Arg6	*	*	*
Arg7	SYSID	CHAR(4)	input

Note: The different uses of Arg5 are shown, because Arg5 is used by the ITEM and NUMITEMS keywords which are alternatives and the argument to the ITEM keyword is an input field when REWRITE is specified.

Table 12. READQ TS: User arguments and associated keywords, data types, and input/output types

Argument	Keyword	Data type	Input/output type
Arg1	QUEUE	CHAR(8)	input
Arg1	QNAME	CHAR(16)	input
Arg2	SET	DATA-AREA, PTR	output
Arg2	INTO	DATA-AREA	output
Arg3	LENGTH	BIN(15)	input/output
Arg4	NUMITEMS	BIN(15)	output
Arg5	ITEM	BIN(15)	input
Arg6	*	*	
Arg7	SYSID	CHAR(4)	input

Table 13. DELETEQ TS: User arguments and associated keywords, data types, and input/output types

Argument	Keyword	Data type	Input/output type
Arg1	QUEUE	CHAR(8)	input
Arg1	QNAME	CHAR(16)	input
Arg2	*	*	*

Table 13. DELETEQ TS: User arguments and associated keywords, data types, and input/output types (continued)

Argument	Keyword	Data type	Input/output type
Arg3	*	*	*
Arg4	*	*	*
Arg5	*	*	*
Arg6	*	*	*
Arg7	SYSID	CHAR(4)	input

Modifying user arguments: User exit programs can modify user arguments, as follows:

For input arguments, the user exit program should obtain sufficient storage to hold the modified argument, set up that storage to the required value, and set the associated pointer in the parameter list to the address of the newly acquired area.

For output arguments, and for input/output arguments, the user exit program can update the argument in place, because the area of storage is represented by a variable in the application which is expected to receive a value from CICS.

Note:

1. CICS does not check changes to argument values, so any changes must be verified by the user exit program making the changes.
2. It is not advisable for XTSEREQ to modify output arguments or for XTSEREQC to modify input arguments.

Adding user arguments:

Global user exit programs can add arguments associated with the SYSID keyword. You must ensure that the arguments you specify or modify in your exit programs are valid.

Assuming that the argument to be added does not already exist, the user exit program must:

1. Obtain storage for the argument to be added
2. Initialize the storage to the required value
3. Select and set up the appropriate pointer from the parameter list
4. Select and set up the appropriate argument existence bit in the EID
5. Modify the parameter list to reflect the new end of list indicator.

Removing user arguments:

User exit programs can remove arguments (for which the program is totally responsible) associated with the SYSID keyword:

Assuming that the argument to be removed exists, the user exit program must:

1. Switch the corresponding argument existence bit to '0'b in the EID
2. Modify the parameter list to reflect the new end of list indicator.

Example program

CICS supplies—as a softcopy listing only (not as a source code file)—an example program, DFH$XTSE, that shows how temporary storage requests can be modified. See Appendix G, "The example program for the XTSEREQ global user exit, DFH$XTSE," on page 917.

Terminal allocation program exit XALCAID

XALCAID is driven when an automatic initiation descriptor (AID) with data is canceled either by the CEMT transaction, running a SET TERMINAL or SET CONNECTION command, or during the reinstallation of a terminal or connection.

XALCAID is invoked only if there is data associated with the AID.

When invoked

Whenever an AID with data is canceled.

Note: It is not possible for the exit to prevent the request from being canceled.

Exit-specific parameters

UEPALTSD

Address of a 4-byte field containing the symbolic identifier of the transaction which was to be started by this request.

UEPALTRM

Address of a 4-byte field containing the identifier of the terminal or connection to which this request was directed.

UEPALDAT

Address of an area of storage containing the data specified in the FROM option; or hexadecimal zeros, in either of the following cases:

- The AID was created by a START request without a FROM option.
- The AID is associated with a channel (in which case the field pointed to by UEPALCHN will be set to a name other than blanks).

UEPALLEN

Address of a fullword binary field containing the length of the FROM data; or hexadecimal zeros, in either of the following cases:

- The AID was created by a START request without a FROM option.
- The AID is associated with a channel (in which case the field pointed to by UEPALCHN will be set to a name other than blanks).

UEPALRQD

Address of an 8-byte field containing the value of the REQID associated with the FROM data. The data was stored in a temporary storage queue with this name. This value was either specified explicitly using the REQID option on the START command, or created internally by CICS.

UEPALQUE
Address of an 8-byte field containing the value specified in the QUEUE option on the START command; or hexadecimal zeros if QUEUE was not specified.

UEPALRTE
Address of a 4-byte field containing the value specified in the RTERMID option on the START command, or hexadecimal zeros if RTERMID was not specified.

UEPALRTA
Address of a 4-byte field containing the value specified in the RTRANSID option on the START command, or hexadecimal zeros if RTRANSID was not specified.

UEPALFMH
Address of a 1-byte field containing the value X'FF' if the data contains FMHs, as specified by the FMH option on the associated START command; or hexadecimal zeros otherwise.

UEPALSTC
Address of a 2-byte field containing the start code. This is "SZ" for FEPI starts; otherwise it is "SD".

UEPALCHN
Address of a 16-byte field containing the name of the channel associated with the AID. If there is no channel associated with the AID, this field is set to blanks.

Return codes

UERCNORM
No other return codes are supplied. The value of the return code is not inspected.

XPI calls
You can use:

- INQ_APPLICATION_DATA
- INQUIRE_SYSTEM

No other XPI calls should be used.

API and SPI commands
No EXEC CICS commands can be used.

Note: The XALTENF exit, used to handle the “terminal not known” condition, is also invoked from the terminal allocation program. XALTENF is described in “‘Terminal not known’ condition exits XALTENF and XICTENF” on page 290.

Related concepts:

“Writing global user exit programs” on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

“Overview of the XPI” on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Terminal control program exits (XTCIN, XTCOUT, XTCATT)

These exits are invoked before I/O events for sequential devices or before task attaches.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XTCIN

When invoked

After an input event for a sequential device.

Exit-specific parameters

UEPTCTTE
Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.

UEPTIOA
Address of the terminal input/output area (TIOA). Your exit program should not change the address. The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are **not** programming interfaces.

UEPTCTLE
Address of the terminal control table line entry (TCTLE). The TCTLE can be mapped using the DSECT DFHTCTLE.

Return codes

UERCNORM
Continue processing.

XPI calls

All can be used. However, note that you cannot use a GETMAIN call to obtain terminal-class storage for use as a replacement TIOA.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XTCOUT

When invoked

Before an output event for a sequential device.

Exit-specific parameters

UEPTCTTE
Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.

UEPTIOA
Address of the terminal input/output area (TIOA). Your exit program should not change the address. The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are **not** programming interfaces.

UEPTCTLE
Address of the terminal control table line entry (TCTLE). The TCTLE can be mapped using the DSECT DFHTCTLE.

Return codes

UERCNORM
Continue processing.

XPI calls
All can be used. However, note that you cannot use a GETMAIN call to obtain terminal-class storage for use as a replacement TIOA.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XTCATT

When invoked
Before task attach.

Exit-specific parameters

UEPTCTTE
Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.

UEPTIOA
Address of the terminal input/output area (TIOA). The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are **not** programming interfaces.

UEPTCTLE
Address of the terminal control table line entry (TCTLE). The TCTLE can be mapped using the DSECT DFHTCTLE.

UEPTRAN
Address of the 4-byte transaction id.

Return codes

UERCNORM
Continue processing.

XPI calls
All can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

'Terminal not known' condition exits XALTENF and XICTENF

The 'terminal not known' condition can occur when intercommunicating CICS regions use both SHIPPABLE terminal definitions and automatic transaction initiation (ATI). The condition is especially likely to arise if autoinstall is used.

SHIPPABLE attribute
Terminals defined with the SHIPPABLE attribute in a terminal-owning region (TOR) do not need a definition in a connected application-owning region (AOR). If necessary to support transaction routing, CICS ships a copy of the definition from the TOR to the AOR. For further information, refer to the *CICS Intercommunication Guide*.

Automatic transaction initiation (ATI)
ATI occurs when an internally generated request leads to the initiation of a transaction. For example, when:

- An application issues an EXEC CICS START command, or
- The transient data trigger level is reached.

Two CICS modules handle ATI requests:

The **interval control program** processes a START command, checks that the terminal is known in the local system, and (when any START time interval elapses) calls the terminal allocation program.

The **terminal allocation program** is called by the interval control program or by the transient data triggering mechanism, and checks that the terminal is known in the local system. If the requested terminal is remote, the terminal allocation program ships an ATI request to the remote system, which initiates transaction routing back to the local system.

For guidance information about ATI, refer to the *CICS Intercommunication Guide*.

'Terminal not known' condition
The 'terminal not known' condition arises when an ATI request is made for a terminal not known in the region. An ATI request can occur in the AOR for a SHIPPABLE terminal before any transaction routing has taken place for the terminal, and so before the definition of the terminal can have been shipped from the TOR to the AOR.

If the 'terminal not known' condition occurs, both the interval control program and the terminal allocation program reject the transaction-initiation request as 'TERMIDERR'.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

The exits

To deal with the 'terminal not known' condition, CICS provides global user exits in the interval control and terminal allocation programs:

XICTENF
In the interval control program

XALTENF
In the terminal allocation program.

CICS drives the XICTENF exit when the 'terminal not known' condition occurs after the interval control program has been invoked by an EXEC CICS START command. CICS drives the XALTENF exit when the 'terminal not known' condition occurs after the terminal allocation program has been invoked by the transient data trigger level or the interval control program. Note that an EXEC CICS START command could result in both exits being invoked.

The exit program must indicate whether the terminal exists on another system and, if so, on which one. CICS passes data to the exit program to help establish this information. You can use the same exit program at both exit points. CICS supplies a sample exit program, DFHXTENF (see Figure 8 on page 297), that can be used at both exits and that can deal unchanged with some typical situations.

The exits are designed to deal with 'terminal not known' conditions that occur in CICS regions other than the TOR. For a TOR/AOR pair, enable the exit program in the AOR. The exits cannot deal with a 'terminal not known' condition in the TOR and the exit program should not normally be enabled there. However, if more than one TOR exists, you may need to enable the exit program in each TOR to deal with requests for terminals owned by other TORs. In this case, the exit program must recognize terminals that should be owned by this system and reject the requests ('UERCTEUN'). Although the exit provides as much data as possible, the logic of your program depends entirely on your system design. A simple solution to the most complex case would be to make the name of each terminal reflect the netname or sysid of its owning region.

Data returned by exit:

The exit program must set a return code in register 15 as follows:

UERCTEUN
Terminal does not exist

UERCNETN
Netname of TOR returned

UERCSYSI
Sysid of TOR returned.

For return codes UERCNETN and UERCSYSI, the exit program must place the netname or sysid of the terminal-owning region in fields UEPxxNTO or UEPxxSYO (where xx is AL or IC).

If the terminal-owning region is a member of a z/OS Communications Server generic resource, the exit program should place the netname of the terminal in

field UEPxxNNO. For information about using ATI with z/OS Communications Server generic resources, see Using ATI with generic resources, in the *CICS Intercommunication Guide*.

Exit XALTENF

Exit XALTENF is invoked by the terminal allocation program when the terminal that an ATI request from transient data or interval control requires is unknown in this system. The exit program is expected to give a return code indicating whether the terminal exists on another connected CICS system and, if so, on which one.

Exit-specific parameters

UEPALEVT
Address of 2 bytes containing the type of request. The equated values of the types are:

UEPALESD
START command with data

UEPALES
START command without data

UEPALETD
Transient data trigger level reached.

UEPALTR
Address of 1 byte containing an indication of whether the task issuing the START command was started by transaction routing. The equated values are:

UEPALTY
A START command was being processed and the task issuing the command was transaction routed to.

UEPALTN
A START command was not being processed **or** a START command was being processed but the task issuing the command was not transaction routed to.

UEPALFS
Address of 1 byte containing an indication of whether the START command was function shipped. The equated values are:

UEPALFY
A START command was being processed and the START was function shipped.

UEPALFN
A START command was not being processed **or** a START was being processed but it was not function shipped.

UEPALTRN
Address of 4 bytes containing the name of the transaction to be run.

UEPALRTR
Address of 4 bytes containing the name of the terminal on which the transaction should run. (If a transient data trigger level was reached and the transient data queue definition specified a system, then this would contain a system identifier.)

UEPALCTR
Address of 4 bytes containing, for START commands, the name of

the current terminal if the command was transaction routed, or the name of the session if the command was function shipped.

For other START commands and for transient data trigger events, the field pointed to contains blanks.

UEPALNTI
Address of 8 bytes containing, for function-shipped START commands, the netname of the last system from which the request came.

For START commands issued in this system by transaction routing to a task, the netname of the last system from which the task was routed.

For other START command situations and for transient data trigger level events, the field pointed to contains blanks.

UEPALSYI
Address of 4 bytes containing, if UEPALNTI contains a netname, the corresponding sysid.

If UEPALNTI does not contain a netname, the field pointed to is blank.

UEPALNTO
Address of 8 bytes containing the contents of UEPALNTI.

If it sets a return code of 'UERCNETN', your exit program must place in this field the netname of the system to which the ATI request should be sent.

UEPALSYO
Address of 4 bytes containing the contents of UEPALSYI.

If it sets a return code of 'UERCSYSI', your exit program must place in this field the sysid of the system to which the ATI request should be sent.

UEPALNNI
Address of a 4-byte input field containing the netname of the terminal on which the transaction is to run, if this is known to CICS. If CICS does not know the netname, the addressed field contains blanks.

UEPALNNO
Address of a 4-byte input/output field containing, on invocation, the contents of UEPALNNI. Your exit program can use this field to supply the netname of the terminal on which the transaction is to run. It is important that your exit program supply a terminal netname if the TOR to which it directs the ATI request is a member of a z/OS Communications Server generic resource.

Return codes

UERCTEUN
Terminal unknown, reject request.

UERCNETN
Terminal known, netname returned in UEPALNTO.

UERCSYSI
Terminal known, sysid returned in UEPALSYO.

XPI calls

You can use:

- INQ_APPLICATION_DATA
- INQUIRE_SYSTEM.

No other XPI calls should be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XICTENF

Exit XICTENF is invoked by the interval control program when the terminal that an EXEC CICS START command requires is unknown in this system.

When invoked

By the interval control program when the terminal that an EXEC CICS START command requires is unknown in this system. The exit program is expected to give a return code indicating whether the terminal exists on another connected CICS system and, if so, on which one.

Exit-specific parameters

UEPICEVT

Address of 2 bytes containing the type of request. The equated values of the types are:

UEPICESD

START command with data

UEPICES

START command without data.

UEPICTR

Address of 1 byte containing an indication of whether the task issuing the START command was started by transaction routing. The equated values are:

UEPICTY

A START command was being processed and the task issuing the command was transaction routed to.

UEPICTN

A START command was not being processed **or** a START command was being processed but the task issuing the command was not transaction routed to.

UEPICFS

Address of 1 byte containing an indication of whether the START command was function shipped. The equated values are:

UEPICFY

A START command was being processed and the START was function shipped.

UEPICFN
A START command was not being processed **or** a START was being processed but it was not function shipped.

UEPICTRN
Address of 4 bytes containing the name of the transaction to be run.

UEPICRTR
Address of 4 bytes containing the name of the terminal on which the transaction should run.

UEPICCTR
Address of 4 bytes containing, for START commands, the name of the current terminal if the command was transaction routed, or the name of the session if the command was function shipped.

For other START commands, the field pointed to contains blanks.

UEPICNTI
Address of 8 bytes containing, for function-shipped START commands, the netname of the last system from which the request came.

For START commands issued in this system by transaction routing to a task, the netname of the last system from which the task was routed.

For other START command situations, the field pointed to contains blanks.

UEPICSYI
Address of 4 bytes containing, if UEPICNTI contains a netname, the corresponding SYSID.

If UEPICNTI does not contain a netname, the field pointed to is blank.

UEPICNTO
Address of 8 bytes containing the contents of UEPICNTI.

If it sets a return code of 'UERCNETN', your exit program must place in this field the netname of the system to which the ATI request should be sent.

UEPICSYO
Address of 4 bytes containing the contents of UEPICSYI.

If it sets a return code of 'UERCSYSI', your exit program must place in this field the sysid of the system to which the ATI request should be sent.

UEPICNNI
Address of a 4-byte input field containing the netname of the terminal on which the transaction is to run, if this is known to CICS. If CICS does not know the netname, the addressed field contains blanks.

UEPICNNO
Address of a 4-byte input/output field containing, on invocation, the contents of UEPICNNI. Your exit program can use this field to supply the netname of the terminal on which the transaction is to run. It is important that your exit program supply a terminal

netname if the TOR to which it directs the ATI request is a member of a z/OS Communications Server generic resource.

Return codes

UERCTEUN
Terminal unknown, reject request.

UERCNETN
Terminal known, netname returned in UEPICNTO.

UERCSYSI
Terminal known, sysid returned in UEPICSYO.

UERCPURG
Task purged during XPI call.

XPI calls

The following must not be used:

- ADD_SUSPEND
- DELETE_SUSPEND
- DEQUEUE
- ENQUEUE
- RESUME
- SUSPEND
- WAIT_MVS.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The sample program for the XALTENF and XICTENF exits, DFHXTENF

DFHXTENF is a sample program that can be used for the XALTENF and XICTENF exits.

One program can be used for both exits, or a separate program can be written for each. Figure 8 on page 297 shows the executable code from the supplied sample program DFHXTENF, which can be used for both exits. DFHXTENF rejects transient data requests, because the action in this case is very much installation-dependent.

```
DFHXTENF CSECT
         DFHVM XTENF
         ENTRY DFHXTENA
DFHXTENA DS    0H
         STM   R14,R12,12(R13)     save registers
         BALR  R11,0               set up base register
         USING *,R11
*
         USING DFHUEPAR,R1         DFHUEH parameter list
*
*        Could check the terminal ID at this point. In this
*        program we assume it is valid. We also choose to accept
*        START requests and reject Transient Data trigger level
*        events.
*
         L     R2,UEPICEVT         access type of request
         CLC   0(2,R2),START       START command?
         BE    STARTCMD            yes
*
         CLC   0(2,R2),STARTDAT    START command with data?
         BNE   NOTSTART            no, must be Transient Data
*
STARTCMD DS    0H
*
*        Accept the default netname if we are Function Shipping.
*        Otherwise build a netname.
*
         L     R2,UEPICFS          access FS information
         CLI   0(R2),UEPICFY       Function Shipping?
         BNE   BLDNETNM            no, build a netname
*
         LH    R15,NETNAME         accept the default netname
         B     EXIT
*BLDNETNM DS    0H
*
*        Build a netname by taking the first character of the
*        terminal ID and appending it to the characters 'CICS'.
*
         L     R2,UEPICNTO         access the output netname field
         L     R3,UEPICRTR         access ID of requested terminal
         MVC   0(8,R2),=C'CICS    '
         MVC   4(1,R2),0(R3)       first character of terminal ID
         LH    R15,NETNAME         netname returned
         B     EXIT
*
NOTSTART DS    0H
         LH    R15,UNKNOWN         reject Transient Data trigger       *
                                   level events
*
EXIT     DS    0H
         L     R14,12(R13)         restore registers except 15
         LM    R0,R12,20(R13)      which contains the return code
         BR    R14
*
*********************************************************************
*        Local constants
*********************************************************************
START    DC    AL2(UEPICES)
STARTDAT DC    AL2(UEPICESD)
NETNAME  DC    AL2(UERCNETN)
UNKNOWN  DC    AL2(UERCTEUN)
*
         DFHEND DFHXTENF
```

Figure 8. Sample program for XALTENF and XICTENF exits

Important

The example in Figure 8 on page 297 is intended purely as a demonstration of some of the possibilities available, and would be impractical in a production environment.

Related concepts:

"Terminal-not-known sample exit program (DFHXTENF)" on page 30
CICS includes a sample global user exit program that handles terminal-not-known conditions arising from START and ATI requests. The sample program is DFHXTENF.

Transaction manager domain exit XXMATT

Exit XXMATT is invoked during transaction attach, and is able to change some of the attributes of the transaction that is being attached.

The exit can change the attach transaction ID of the transaction by changing the field addressed by UEPATPTI. You cannot use **EXEC CICS** commands from this exit.

Exit-specific parameters

UEPTRANID
The address of transaction ID (see Notes).

UEPUSER
The address of the user ID associated with the transaction if the current task is a user task (see Notes).

UEPTERM
The address of the terminal ID associated with the transaction, if any (see Notes).

UEPPROG
The address of the application program name for this transaction, if any (see Notes).

UEPATPTI
The address of a 4 byte field containing the primary transaction ID. You can change the primary transaction ID by modifying the addressed field.

UEPATOTI
The address of the 4 byte attach transaction ID. A transid of X'00000000' indicates that a transid was not supplied on the attach.

UEPATTPL
The address of an area containing the length of the attach TPName. A length of zero indicates that a TPName was not supplied on the attach.

UEPATTPA
The address of a fullword containing the address of the attach TPName. The attach TPName can be 1 through 64 bytes long, as defined by UEPTTPL.

UEPATLOC
The address of a 1 byte field indicating whether the transaction was found. Note that if the transaction was not found but system initialization parameters DTRTRAN and DTRPGM are specified, the transaction specified on DTRTRAN is attached, and CICS considers that the transaction has been found.

Equated values are:

UEATFND
The transaction was found.

UEATNFND
The transaction was not found.

UEPATTST
The address of a 1 byte transaction definition state. Equated values for the definition state are:

UEATENAB
The transaction is enabled.

UEATDISA
The transaction is disabled.

UEPATTTK
The address of a doubleword containing a transaction token. Note that some of the transaction manager XPI calls require this token to identify the transaction that is being attached.

Return codes

UERCNORM
Continue attach processing.

XPI calls

The user exit can inquire on the transaction being attached, using the UEPATTTK transaction token as input to the XMIQ INQUIRE_TRANSACTION XPI call.

The exit can also set the total priority and TCLASS, using the XMIQ SET_TRANSACTION XPI call.

Most of the XPI calls can be used, but with caution since typically this exit is invoked under the TCP task. Thus it is advisable not to issue any XPI calls that might cause the TCP task to wait.

Note:

1. The following XPI calls can be useful for obtaining information that could be used to modify the attach of a transaction:
   - INQUIRE_TRANSACTION
   - INQUIRE_MXT
   - INQUIRE_TCLASS
   - INQUIRE_TRANDEF
   - INQUIRE_SYSTEM
2. The fields UEPTRANID, UEPUSER, UEPTERM, and UEPPROG are common to many of the domain global user exit points, and normally return values associated with the current user task. In the case of XXMATT, however, the user task that is being attached is **not** the current task when the exit is invoked. Until task attach is complete, the current task is the CICS task that is performing the attach.

   When the task being attached is for a task started by an immediate START command; that is, a START without an interval, the current task is the task that issues the START command, and the fields contain values associated with that task.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Transient data program exits (XTDREQ, XTDIN, XTDOUT)

The XTDREQ exit intercepts a transient data request before request analysis. The XTDOUT and XTDIN exits are invoked before and after data is exchanged with QSAM or VSAM.

The CICS transient data facility is threadsafe, so CICS can process transient data requests on an open TCB. Transient data requests are also threadsafe when you function ship them to a remote region over an IPIC connection. To optimize TCB switching and gain the performance benefits of the open transaction environment, programs running at XTDREQ, XTDIN, and XTDOUT must be coded to threadsafe standards and defined to CICS as threadsafe.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XTDREQ

Exit XTDREQ is invoked before request analysis.

Exit-specific parameters

UEPTDQUE
Address of 4-byte TD queue name.

UEPTDTYP
Address of 1-byte TD request type. Values are:

UEPTDPUT
PUT request

UEPTDGET
GET request

UEPTDPUR
PURGE request.

Return codes

UERCNORM
Continue TD processing.

UERCTDOK
Quit TD processing – returning 'NORMAL' to the caller.

UERCTDNA
Quit TD processing – returning 'NOTAUTH' to the caller.

UERCPURG
Task purged during XPI call.

XPI calls
You can use:

- INQ_APPLICATION_DATA
- INQUIRE_SYSTEM
- WAIT_MVS

Do not use any other calls.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XTDIN

Exit XTDIN is invoked after CICS receives data from QSAM (for extrapartition) or VSAM (for intrapartition).

Exit-specific parameters

UEPTDQUE
Address of the 4-byte TD queue name.

UEPTDAUD
Address of the unmodified TD data.

UEPTDLUD
Address of the fullword length of the unmodified TD data.

UEPTDAMD
Address of the TD data modified by the exit program.

UEPTDLMD
Address of the fullword length of the TD data modified by the exit program.

Return codes

UERCNORM
Continue TD processing.

UERCPURG
Task purged during XPI call.

XPI calls
You can use:

- INQ_APPLICATION_DATA
- INQUIRE_SYSTEM
- WAIT_MVS

Do not use any other calls.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XTDOUT

Exit XTDOUT is invoked before CICS passes data to a QSAM (for extrapartition) or VSAM (for intrapartition) user-defined transient data queue.

Exit-specific parameters

UEPTDQUE
Address of the 4-byte TD queue name.

UEPTDAUD
Address of the unmodified TD data.

UEPTDLUD
Address of the fullword length of the unmodified TD data.

UEPTDAMD
Address of the TD data modified by the exit program.

UEPTDLMD
Address of the fullword length of TD data modified by the exit program.

UEPTDNUM
Address of the fullword containing the number of items in the list.

UEPTDCUR
Address of the fullword containing the number of the current item.

Return codes

UERCNORM
Continue TD processing.

UERCTDOK
Quit TD processing – returning 'NORMAL' to the caller.

Note: If you return UERCTDOK to suppress the first line of a multiline message, the rest of the message is not presented to XTDOUT, but is also suppressed.

UERCPURG
Task purged during XPI call.

XPI calls

You can use:

- INQ_APPLICATION_DATA
- INQUIRE_SYSTEM
- WAIT_MVS

Do not use any other calls.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Transient data EXEC interface program exits XTDEREQ and XTDEREQC

The XTDEREQ exit intercepts a transient data request before any action has been taken on it by transient data. The XTDEREQC exit intercepts a transient data request after transient data has completed its processing.

You can change the XTDEREQ exit to perform the following tasks:

- Analyze the request to determine its type, the keywords specified, and their values.
- Modify any value specified by the request before the command is executed.
- Set return codes to specify either of the following instructions:
  - CICS should continue with the request, with any modifications that you made.
  - CICS should bypass the request. If you set this return code, you must also set up return codes for the EXEC interface block (EIB), as if you had processed the request yourself.

You can change the XTDEREQC exit to perform the following tasks:

- Analyze the request, to determine its type, the keywords specified, and their values.
- Set return codes for the EIB.

The CICS transient data facility is threadsafe, so CICS can process transient data requests on an open TCB. Transient data requests are also threadsafe when you function ship them to a remote region over an IPIC connection. To optimize TCB switching and gain the performance benefits of the open transaction environment, programs that run at XTDEREQ and XTDEREQC must be coded to threadsafe standards and defined to CICS as threadsafe.

Both exits are passed eight parameters as follows:

- The address of the command-level parameter structure.
- The address of a token (UEPTDTOK) used to pass 4 bytes of data from XTDEREQ to XTDEREQC.
- The addresses of copies of four pieces of return code and resource information from the EIB.
- The address of a token (UEPTSTOK) that is valid throughout the life of a task.
- The address of an exit recursion count (UEPRECUR).

Example program

CICS supplies, as a softcopy listing only and not as a source code file, an example program, DFH$XTSE, that shows how to modify fields in the command-level

parameter structure passed to EXEC interface exits. DFH$XTSE is listed in Appendix G, “The example program for the XTSEREQ global user exit, DFH$XTSE,” on page 917.

Related concepts:

“Writing global user exit programs” on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XTDEREQ

Exit XTDEREQ is invoked before CICS processes a transient data API request.

Exit-specific parameters

UEPCLPS
Address of the command-level parameter structure. See “The UEPCLPS exit-specific parameter” on page 308.

UEPTDTOK
Address of the 4-byte token to be passed to XTDEREQC. UEPTDTOK allows you, for example, to pass a work area to exit XTDEREQC.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code ‘EIBRCODE'. For details of EIB return codes, refer to EIB fields, in the *CICS Application Programming Reference* manual.

UEPRESP
Address of a 4-byte binary copy of the EIB response code ‘EIBRESP'.

UEPRESP2
Address of a 4-byte binary copy of the EIB response code ‘EIBRESP2'.

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See “Using the task token UEPTSTOK” on page 19.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

Return codes

UERCNORM
Continue processing.

UERCBYP
The transient data EXEC interface program ignores this request.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.

API and SPI commands

All can be used, except for:

EXEC CICS SHUTDOWN
EXEC CICS XCTL

Note: Take care when issuing recursive commands. For example, you must avoid entering a loop when issuing a transient data request from the XTDEREQ exit. Use of the recursion counter UEPRECUR is recommended.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XTDEREQC

Exit XTDEREQC is invoked after a transient data API request has completed, and before return from the transient data EXEC interface program.

Exit-specific parameters

UEPCLPS
Address of the command-level parameter structure. See "The UEPCLPS exit-specific parameter" on page 308.

UEPTDTOK
Address of the 4 byte token to be passed to XTDEREQC. This allows you, for example, to pass a work area to exit XTDEREQC.

UEPRCODE
Address of a 6-byte hexadecimal copy of the EIB return code 'EIBRCODE'. For details of EIB return codes, refer to EIB fields, in the *CICS Application Programming Reference* manual.

UEPRESP
Address of a 4-byte binary copy of the EIB response code 'EIBRESP'.

UEPRESP2
Address of a 4-byte binary copy of the EIB response code 'EIBRESP2'.

UEPTSTOK
Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 19.

UEPRECUR
Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

UEPRSRCE
Address of an 8-character copy of the EIB resource value, EIBRSRCE.

UEP_TD_REMOTE_SYSTEM
If the request is to be sent to a remote region, is the address of an area containing the 4-byte name of the remote region. (The remote region may have been specified by, for example, the SYSID option of the command, function shipping, or the REMOTESYSTEM option of the TDQUEUE definition.)

If the request is to be executed on the local region, this parameter is the address of a 4-byte area containing blanks.

UEP_TD_REMOTE_NAME
If the request is to be sent to a remote region, is the address of an area containing the 4-character name by which the queue is known in the remote region.

Return codes

UERCNORM
Continue processing.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.

API and SPI commands
All can be used, except for:
EXEC CICS SHUTDOWN
EXEC CICS XCTL

Note: Take care when issuing recursive commands. For example, you must avoid entering a loop when issuing a transient data request from the XTDEREQC exit. Use of the recursion counter UEPRECUR is recommended.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

The command-level parameter structure

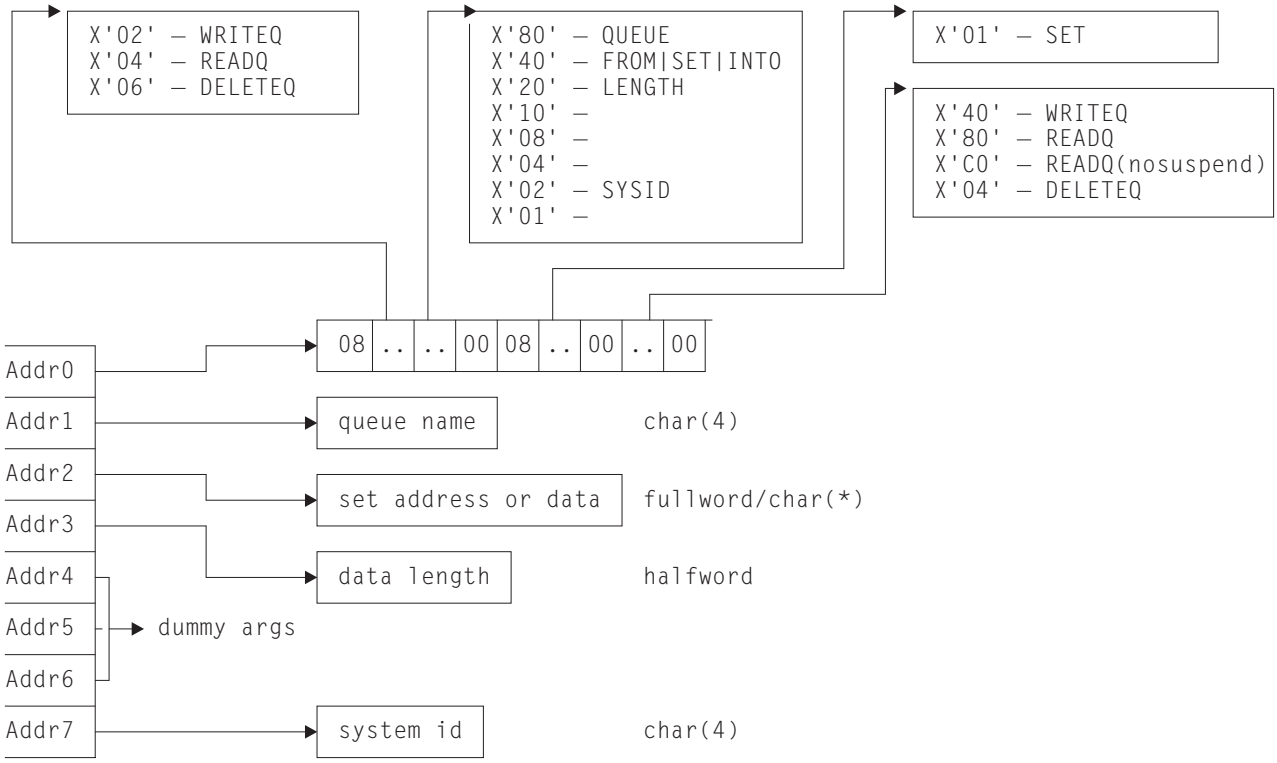


Figure 9. The command-level parameter structure for transient data

The command-level parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of an 8-byte area that describes the type of request and identifies each keyword specified with the request. The remaining addresses point to pieces of data associated with the request. (For example, the second address points to the queue name.)

You can examine the EID to determine the type of request and the keywords specified. You can examine the other parameters in the list to determine the values of the keywords. You can also modify values of keywords specified on the request. (For example, you could change the sysid specified in the request.)

End of parameter list indicator

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

For example, if the parameter list specifies only the first two addresses (TD_ADDR0, the address of the EID, and TD_ADDR1, the address of the name of the queue named in a DELETEQ request), the high-order bit is set on in TD_ADDR1. If you extend the parameter list by setting the address of a SYSID in TD_ADDR7, you must reset the high-order bit in TD_ADDR1 and set it on in TD_ADDR7 instead.

The maximum size of parameter list is supplied to the exit, thus allowing your exit program to add any parameters not already specified without needing to first obtain more storage.

The original parameter list, as it was before XTDEREQ was invoked, is restored after the completion of XTDEREQC. It follows that the execution diagnostic facility (EDF) displays the original command before **and** after execution. **EDF does not display any changes made by the exit**.

The UEPCLPS exit-specific parameter:

The UEPCLPS exit-specific parameter is included in both exit XTDEREQ and exit XTDEREQC. It contains the address of the command-level parameter structure.

The command-level parameter structure contains 8 addresses, TD_ADDR0 through TD_ADDR7. It is defined in the DSECT TD_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHTDUED.

The command-level parameter list is made up as follows:

TD_ADDR0
is the address of an 8-byte area called the EID, which is made up as follows:

- **TD_GROUP**
- **TD_FUNCT**
- **TD_BITS1**
- **TD_BITS2**
- **TD_EIDOPT5**
- **TD_EIDOPT6**
- **TD_EIDOPT7**

TD_GROUP
Always X'08', indicating that this is a transient data request.

TD_FUNCT
One byte that defines the type of request:

X'02' WRITEQ

X'04' READQ

X'06' DELETEQ.

TD_BITS1
Existence bits that define which arguments were specified. To obtain the argument associated with a keyword, you need to use the appropriate address from the command-level parameter structure. Before using this address, you must check the associated existence bit. If the existence bit is set off, the argument was not specified in the request and the address should not be used.

X'80' Set if the request contains an argument for the QUEUE keyword. If set, **TD_ADDR1** is meaningful.

X'40' Set if the request contains an argument for any of the INTO, SET, or FROM keywords. If set, **TD_ADDR2** is meaningful.

X'20' Set if the request contains an argument for the LENGTH keyword. If set, **TD_ADDR3** is meaningful.

X'02' Set if the request contains an argument for the SYSID keyword. If set, **TD_ADDR7** is meaningful.

TD_BITS2
Two bytes not used by transient data.

TD_EIDOPT5
Indicates whether certain keywords were specified on the request.

X'01' SET (and not INTO) was specified.

TD_EIDOPT6
One byte not used by transient data.

TD_EIDOPT7
Indicates whether certain functions, keywords or both were specified on the request:

X'40' WRITEQ specified

X'80' READQ specified

X'C0' READQ(nosuspend) specified

X'04' DELETEQ specified.

TD_ADDR1
is the address of a 4-byte area containing the name from QUEUE.

TD_ADDR2
is the address of one of the following:

- A 4-byte address from SET (if the request is READQ and **TD_EIDOPT5** indicates that this is SET).
- Data from INTO (if the request is READQ and **TD_EIDOPT5** indicates that this is not SET). You cannot modify this bit in your user exit.
- Data from FROM (if the request is WRITEQ).

TD_ADDR3
is the address of one of the following:

- The halfword value of LENGTH (if the request is READQ or WRITEQ). **Warning:** For requests that specify INTO, do not change the value of LENGTH to a value greater than that specified by the application. To do so causes a storage overlay in the application.

TD_ADDR4
is the address of a value intended for CICS internal use only. It must not be used.

TD_ADDR5
is the address of a value intended for CICS internal use only. It must not be used.

TD_ADDR6
is the address of a value intended for CICS internal use only. It must not be used.

TD_ADDR7
is the address of an area containing the value of SYSID.

TD_ADDR8
is the address of a value intended for CICS internal use only. It must not be used.

Modifying fields in the command-level parameter structure:

Some fields that are passed to transient data are used as input to the request, some are used as output fields, and some are used for both input and output. The method your user exit program uses to modify a field depends on the usage of the field.

The following are always input fields:

- QUEUE
- FROM
- SYSID

The following are always output fields:

- INTO
- SET

LENGTH is an input field on a WRITEQ request, and an output field on a READQ request that specifies SET. It is both an input and an output field on a READQ request that specifies INTO.

Modifying input fields:

The correct method of modifying an input field is to create a new copy of it, and to change the address in the command-level parameter list to point to your new data.

Note: You must never modify an input field by altering the data that is pointed to by the command-level parameter list. To do so would corrupt storage belonging to the application program and would cause a failure when the program attempted to reuse the field.

Modifying output fields: The technique described in "Modifying input fields" is not suitable for modifying output fields. (The results would be returned to the new area instead of the application's area, and would be invisible to the application.)

An output field is modified by altering the data that is pointed to by the command-level parameter list. In the case of an output field, you can modify the application's data in place, because the application is expecting the field to be modified.

Modifying fields used for both input and output:

An example of a field that is used for both input and output is LENGTH on a READQ request that specifies INTO. You can treat such fields in the same way as output fields, and they are considered to be the same.

Modifying the EID:

It is not possible to modify the EID to make major changes to requests, such as changing a READQ request to a WRITEQ request. However, you can make minor changes to requests, such as turning on the existence bit for SYSID so that the request can be changed into one that is shipped to a remote system.

The list that follows shows the bits in the EID that can be modified. Any attempt to modify any other part of the EID is ignored.

TD_BITS1

X'20' The existence bit for LENGTH.

X'02' The existence bit for SYSID.

TD_EIDOPT5

X'01' Existence bit for SET keyword. You cannot modify this bit from your user exit.

TD_EIDOPT7

Changes to TD_EIDOPT7 are limited to READQ requests. X'80'-READQ is interchangeable with X'C0'-READQ(nosuspend). No other changes may be made to this byte.

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the transient data request only.

Note: Your user exit program is prevented from making major changes to the EID. However, you must take great care when making the minor modifications that **are** permitted.

The EIB:
Copies of EIBRSRCE, EIBRCODE, EIBRESP, and EIBRESP2 are passed to the exit, so that you can:

- Modify or set completion and resource information in XTDEREQ and XTDEREQC
- Examine completion and resource information in XTDEREQC.

You can update the copies of EIBRSRCE, EIBRCODE, EIBRESP, and EIBRESP2 that you are given in the parameter list. Transient data copies your values into the real EIB after the completion of XTDEREQC; or if you specify a return code of ' bypass' in XTDEREQ.

You must set valid transient data responses. You must set all three of EIBRCODE, EIBRESP, and EIBRESP2 to a consistent set of values, such as would be set by CICS transient data to describe a valid completion. **CICS does not police the consistency of EIBRCODE, EIBRESP, and EIBRESP2**. However, if EIBRCODE is set to a non-zero value and EIBRESP is set to zero then CICS will override EIBRESP with a non-zero value. To aid you in setting the values of EIBRCODE, EIBRESP, and EIBRESP2, the values used by transient data are specified in DFHTDUED.

User log record recovery program exits XRCINIT and XRCINPT

At warm and emergency restart, updates made to recoverable CICS resources that were not committed when the system terminated must be backed out. XRCINIT and XRCINPT are invoked from the user log record recovery program, which is used to back out, where necessary, user-written system log entries.

XRCINIT is invoked at warm and emergency restart:

- Before the first user recovery record is delivered to XRCINPT
- When all such records have been delivered to XRCINPT.

XRCINPT is invoked whenever a user log record is read from the system log.

You can use XRCINPT to change the default actions taken by CICS at emergency restart for particular user-journaled records. Records passed to XRCINPT are those in UOWs that:

- Appeared in the last complete activity keypoint
- Were in flight when CICS terminated

- Committed, backed out, or went indoubt after the start of the last complete activity keypoint. (However, this only applies to those records for which the leftmost bit of the JTYPEID specified in the WRITE JOURNALNAME(DFHLOG) request was a one.)

Records written by the activity keypoint exit XAKUSER are passed only if they appear in the last complete activity keypoint. They are passed after all other records. The order of presentation of records may therefore be different from their order in the reverse log stream sequence.

The format of records passed to the exit is:

Offset	Field contents
0	JTYPEID
2	Reserved
4	Length of prefix data (L). (Zero if no prefix)
8	Prefix data (if any)
8 + L	Log data

The record is mapped by the DSECT CL_USER_HEADER in copybook DFHLGGFD.

When using XRCINIT and XRCINPT, you should bear in mind that the exits may be invoked before recovery of temporary storage and transient data resources is complete.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Coding the exit programs

You can use CICS services in exit programs invoked from these exits using the XPI or EXEC CICS commands.

You need to consider the following:

- There is a restriction on using the XPI early during initialization: do not invoke exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR and INQUIRE_MONITOR_DATA until the second phase of the PLTPI.
- There are also restrictions on the use of EXEC CICS commands in these exits:
  - You cannot use EXEC CICS commands to access terminal control services.
  - You are strongly advised not to use temporary storage, transient data, file control, journal control, or DL/I services, because the resources that you try to access may also be in a state of recovery and therefore "not open for business". Attempting to access resources in these circumstances causes, at best, serialization of the recovery tasks and, at worst, a deadlock.

    If you do issue file control requests in programs invoked from these exits, note that:

    - If an exit program acquires an area as a result of a file control request, it is the responsibility of the program to release that area.

- An exit program must not attempt to make any file control requests to a file referring to a VSAM data set with a string number of 1, unless no action is specified for that file during the initialization exit.

– Your exit program must not issue EXEC CICS commands if the recovery is as the result of an EXEC CICS SYNCPOINT ROLLBACK request.
– Exit programs that issue EXEC CICS commands must first address the EIB. See "Using CICS services" on page 4.
– Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See "Returning values to CICS" on page 10.
– Exit programs invoked from these exits must be translated with the NOEDF option, if they issue EXEC CICS commands. See "EDF and global user exits" on page 6.

- Task-chained storage acquired in an exit program is released at the completion of emergency restart processing. However, the exit program should attempt to release the storage as soon as its contents are no longer needed.
- No exit program should reset either the absent or no-action indicators set by the file control backout program.
- Take care when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when an RC request is issued from these exits.

Enabling the exit programs

To enable these exits, you must do one of the following:

- Specify the system initialization parameter TBEXITS=(name1,name2,name3,name4,name5,name6), where name1 through name6 are the names of your user exit programs for XRCINIT, XRCINPT, XFCBFAIL, XFCLDEL, XFCBOVER, and XFCBOUT.
- Enable the exits during the first stage of initialization using a PLTPI program.

If you use the TBEXITS parameter to enable the exits, a global work area of 4 bytes is provided. If you use a PLTPI program, you can select the size of the global work area. You can also enable more than one exit program for use at each exit point; the TBEXITS parameter allows only one exit program at each exit point. PLTPI processing is described in Chapter 5, "Writing initialization and shutdown programs," on page 409.

Exit XRCINIT

When invoked

At warm and emergency restart:

- Before the first user recovery record is delivered to XRCINPT
- When all such records have been delivered to XRCINPT.

Exit-specific parameters

UEPTREQ

Address of a 1-byte flag indicating the reason for the call. When UEPTREQ has a value of UEUSINIT, the exit has been invoked at the start of user recovery, and when UEPTREQ has a value of UEUSTERM, the exit has been invoked at the end of user recovery.

UEPRSTRT

Address of a 1-byte flag that indicates how CICS was restarted:

UEPRWARM
Warm start

UEPREMER
Emergency start.

Return codes

UERCNORM
Continue processing. No other return codes are supported.

XPI calls
All can be used. See "User log record recovery program exits XRCINIT and XRCINPT" on page 311 for restrictions.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XRCINPT

When invoked
At warm and emergency restart, once for each user log record found in the system log.

Exit-specific parameters

UEPUOWST
Address of a 1-byte flag indicating the disposition of the unit of work. The possible values are:

UEPUOWAK
Activity keypoint record

UEPUOWCM
Unit of work committed

UEPUOWBO
Unit of work backed out

UEPUOWIF
Unit of work was in-flight

UEPUOWID
Unit of work was indoubt.

UEPLGREC
Address of the log record just read. The journal control record can be mapped using the information supplied in ../../com.ibm.cics.ts.performance.doc/topics/dfht34q.dita.

UEPLGLEN
Address of a fullword containing the length of the log record.

UEPTAID
Address of a 4-byte field containing the task identifier.

UEPTRID
Address of a 4-byte field containing the transaction identifier.

UEPTEID
Address of a 4-byte field containing the terminal identifier.

Note: The values of the fields addressed by UEPTAID, UEPTRID, and UEPTEID are meaningless for activity keypoint records (that is, if the field addressed by UEPUOWST contains UEPUOWAK).

Return codes

UERCNORM
Continue processing.

UERCBYP
Bypass this record.

XPI calls
All can be used. See "User log record recovery program exits XRCINIT and XRCINPT" on page 311 for restrictions.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

SNA LU management program exit (XZCATT)

This exit is invoked before a task attach for a LU terminal task.

When invoked
Before task attach for terminal tasks.

Exit-specific parameters

UEPTCTTE
Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.

UEPTIOA
Address of the terminal input/output area (TIOA). The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are **not** programming interfaces.

UEPTPN
Address of the APPC transaction process name (TPN), or the LU6.1 process name (DPN), whose length is addressed by the parameter UEPTPNL.

UEPTPNL
Address of a 1-byte field containing the length of the TPN or DPN.

UEPTRAN
Address of the 4-byte transaction ID.

Note: The exit program must not change the TRANSID of tasks started by automatic transaction initiation (ATI). (This is because CICS needs to match the TRANSID in its program control table with the TRANSID in the automatic initiate descriptor (AID) that was created in the AOR.)

Return codes

UERCNORM
Continue processing.

XPI calls
All can be used.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

SNA working-set module exits (XZCIN, XZCOUT, XZCOUT1, and XZIQUE)

These exits are invoked after I/O events or before messages are disassembled into request units (RUs).

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XZCIN

This exit is invoked after an input event.

When invoked
After an input event.

Exit-specific parameters

UEPTCTTE
Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.

UEPTIOA
Address of the terminal input/output area (TIOA). Your exit program should not change the address. The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces.

Return codes

UERCNORM
Continue processing.

XPI calls
All can be used. However, do not use a GETMAIN call to obtain terminal-class storage for use as a replacement TIOA. This is because there are several internal pointers to the TIOA, and if any one of these is not updated the application might experience problems.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XZCOUT

This exit is invoked before an output event.

When invoked
Before an output event.

Exit-specific parameters

UEPTCTTE
Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.

UEPTIOA
Address of the terminal input/output area (TIOA). Your exit program should not change the address. The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are **not** programming interfaces.

Note: In certain circumstances—for example, when XZCOUT is invoked before the send of a NULL RU—UEPTIOA contains zeroes.

Return codes

UERCNORM
Continue processing.

XPI calls
All can be used. However, we do not recommend that you use a GETMAIN call to obtain terminal-class storage for use as a replacement TIOA. This is because there are several internal pointers to the TIOA, and if any one of these is not updated the application may experience problems.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Exit XZCOUT1

This exit is invoked before a message is deconstructed into RUs.

When invoked

Before a message is broken into RUs.

Exit-specific parameters

UEPTCTTE

Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.

UEPTIOA

Address of the terminal input/output area (TIOA). Your exit program should not change the address. The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are **not** programming interfaces.

Return codes

UERCNORM

Continue processing.

XPI calls

All can be used. However, we do not recommend that you use a GETMAIN call to obtain terminal-class storage for use as a replacement TIOA. This is because there are several internal pointers to the TIOA, and if any one of these is not updated the application may experience problems.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

XZIQUE exit for managing MRO and APPC intersystem queues

You can use the XZIQUE exit to control the number of queued requests for sessions on MRO and APPC connections.

Note:

- Queued requests for sessions are known as "*allocate queues*".
- The equivalent global user exit to control the number of queued requests for sessions on IP interconnectivity (IPIC) connections is XISQUE: see "XISQUE exit for managing IPIC intersystem queues" on page 327.
- There are several methods that you can use to control the length of intersystem queues. For a description of the various methods, see the *CICS Intercommunication Guide*.

The XZIQUE exit enables you detect queuing problems (bottlenecks) early. It extends the function provided by the XISCONA global user exit, that is described in "Intersystem communication program exits, XISCONA, XISLCLQ, and XISQLCL" on page 171, which is invoked only for function shipping and DPL requests. XZIQUE is invoked for transaction routing, asynchronous processing, and distributed transaction processing requests, as well as for function shipping and DPL. Compared with XISCONA, it receives more detailed information on which to base its decisions.

XZIQUE enables allocate requests to be queued or rejected, depending on the length of the queue. It also allows a connection on which there is a bottleneck to be terminated and then re-established.

Interaction with the XISCONA exit:

There is no interaction between the XZIQUE and XISCONA global user exits. If you enable both exits, XISCONA and XZIQUE could both be invoked for function shipping and DPL requests, although this is not recommended.

Therefore, you should ensure that only one of these exits is enabled. Because of it provides more function and greater flexibility, it is recommended that you use XZIQUE rather than XISCONA.

If you already have an XISCONA global user exit program, you could possibly modify it for use at the XZIQUE exit point.

When the XZIQUE exit is invoked:
The XZIQUE global user exit is invoked, if it is enabled, at the following times:

- Whenever CICS tries to acquire a session with a remote system and there is no free session available. It is invoked whether or not you have specified the QUEUELIMIT option on the CONNECTION definition, and whether or not the limit has been exceeded. It is not invoked if the allocate request specifies NOQUEUE or NOSUSPEND.

  Requests for sessions can arise in a number of ways, such as explicit EXEC CICS ALLOCATE commands issued by DTP programs, or by transaction routing or function shipping requests.
- Whenever an allocate request succeeds in finding a free session, after the queue on the connection has been purged by a previous invocation of the exit program. In this case, your exit program can indicate that CICS is to continue processing normally, resuming queuing when necessary.

Using an XZIQUE global user exit program:

When the exit is enabled, your XZIQUE global user exit program is able to check on the state of the allocate queue for a particular connection in the local system.

Information is passed to the exit program in a parameter list, that is structured to provide data about non-specific allocate requests, or requests for specific modegroups, depending on the session request. Non-specific allocate requests are for MRO, LU6.1, and APPC sessions that do not specify a modegroup.

Using the information passed in the parameter list, your global user exit program can decide (based on queue length, for example) whether CICS is to queue the allocate request. Your program communicates its decision to CICS by means of one of the return codes CICS provides. These are:

UERCAQUE

This return code indicates that CICS is to queue the allocate request.

The total number of allocate requests queued against the connection is provided in field A14ESTAQ of the system entry statistics (for all non-specific allocates) or A20ESTAQ of the mode entry statistics (for specific modegroup allocates). See DSECTs DFHA14DS or DFHA20DS for details. CICS passes to the exit program, in the exit specific parameter UEPQUELIM, the QUEUELIMIT parameter from the connection definition.

If the limit has not been reached, you can return control to CICS with return code UERCAQUE.

UERCAPUR

This return code indicates that CICS is to reject the allocate request and return SYSIDERR to the application program, but leave the existing queue unchanged.

If the number of queued allocate requests has reached the limit set on the QUEUELIMIT parameter for the connection, you can request that CICS rejects the request. However, you should first check whether the state of the link is satisfactory. This means checking that the rate of allocation of sessions is acceptable. Use the time the queue was started, the current time, and the total number of allocates processed since the queue began, to determine the rate at which CICS is processing requests. The relevant fields are: UEPSAQTS and UEPSACNT for non-specific allocate requests; and UEPMAQTS and UEPMACNT for specific modegroup requests.

To determine whether CICS is allocating requests for sessions on this connection at an acceptable rate, you can compare the calculated time with either of the following:

1. The parameter from the connection definition, MAXQTIME, which is passed in the exit specific parameter UEPEMXQT
2. Some other preset time value.

If the processing time using this kind of formula is acceptable, return control to CICS with return code UERCAPUR to purge only this request.

UERCAKLL or UERCAKLM

These return codes indicate that you want CICS to deal with the request as follows:

- UERCAKLL—reject this request, purge all other queued allocate requests on this connection, and send an information message to the operator console.
- UERCAKLM—reject this request, purge all other queued modegroup allocate requests on this connection, and send an information message to the operator console.

If the queue limit has been reached but the performance of allocate processing against the queue is below the acceptable limits defined in your user exit program, you can return control to CICS as follows:

- For non-specific allocate requests, use return code UERCAKLL. UERCAKLL also returns SYSIDERR to all application programs waiting on the purged allocate requests. CICS sets the UEPFLAG parameter to UEPRC8 on subsequent calls to your XZIQUE exit program to indicate that UERCAKLL was returned previously to purge the queue.
- For specific modegroup allocate requests, use return code UERCAKLM. UERCAKLM also returns SYSIDERR to all application programs waiting on the purged allocate requests. CICS sets the UEPFLAG parameter to

UEPRC12 on subsequent calls to your XZIQUE exit program to indicate that UERCAKLM was returned previously to purge the queue.

Purging a queue that is causing congestion in the flow of tasks frees task slots that are needed to prevent the system becoming clogged. The more you allow a session queue to grow, the more likely you are to reach the task ceiling set by the MAXT parameter, and then cause a queue of incoming tasks in the local region that cannot be attached. Note that some internal CICS requests (such as those for the LU services model transactions CLS1, CLS2, and CLS3) are not purged by return codes UERCAKLL and UERCAKLM.

If a queue has been purged previously (with UERCAKLL or UERCAKLM) but there are no queued requests currently, check the number of successful allocates since the queue was last purged. For non-specific allocate requests, this number is in UEPSARC8, and for specific modegroup requests, this number is in UEPMAR12. If no requests of this type have been allocated on this connection since the queue was last purged, the problem that caused the purge previously has not been resolved, and this request should be rejected with UERCAPUR.

If the UEPSARC8 or UEPMAR12 parameters show that allocates are being processed, you should use UERCAQUE to resume queuing of requests. If you return with UERCAQUE in this case, CICS issues an information message to the console to signal that queuing has been resumed.

Note: The address of the system entry statistics record, UEPCONST, is supplied for both non-specific and specific modegroup allocate requests.

The address of the modegroup statistics record, UEPMODST, is set to zeros for non-specific allocate requests. This address is supplied only if the request is for a specific modegroup.

If the exit is invoked after a successful allocate following the suppression of queuing, you can use the following return code:

UERCNORM
This return code indicates that CICS is to resume normal processing on the link, including queuing of requests.

Statistics fields in DFHA14DS and DFHA20DS:

There are some statistics fields that your XZIQUE global user exit program can use to control queues.

A14EALRJ:

Each time an XZIQUE global user exit program returns with a request to reject a request, CICS increments field A14EALRJ in the system entry connection statistics

Field A14EALRJ (allocate rejected) is in DSECT DFHA14DS and is provided to help you to tune the queue limit. Normally, if the number of sessions and the queue limit defined for a link are correctly balanced, and there has been no abnormal congestion on the link, the A14EALRJ should be zero. If the rejected allocates field is non-zero it probably indicates that some action is needed.

A14EQPCT and A20EQPCT:

Each time an XZIQUE global user exit program returns with a request to purge a queue, CICS increments a new field in either the system entry connection statistics (field A14EQPCT) or mode entry connection statistics (field A20EQPCT).

A14EQPCT
The count of the number of times the queue has been purged for the connection as a whole.

A20EQPCT
The count of the number of times the mode group queue has been purged.

For detailed information about statistics fields, what they contain and how they are updated, see ISC/IRC system entry: Resource statistics, in the *CICS Performance Guide*.

Exit XZIQUE:

Exit XZIQUE is invoked when an allocate request for a session is about to be queued, and when an allocate request succeeds following previous suppression of queuing.

When invoked
Whenever:

1. An allocate request for a session is about to be queued
2. An allocate request succeeds following previous suppression of queuing.

Exit-specific parameters

UEPZDATA
Address of the 70-byte area containing the information listed. This area is mapped by the DSECT in copybook DFHXZIDS.

Area addressed by UEPZDATA

UEPSYSID
The 4-byte SYSID of the connection.

UEPREQ
A 2-byte origin-of-request code, which can have the following values:

TR Transaction routing

FS Function shipping (includes distributed program link)

AL Other kinds of intercommunication (for example, distributed transaction processing (DTP) or CPI Communications).

UEPREQTR
The 4-byte identifier of the requesting transaction (applicable only when the origin-of-request code is FS or AL).

UEPTRAN
The 4-byte identifier of the transaction being routed (applicable only when origin of request is TR).

UEPFLAG
A 1-byte flag indicating whether a return code 8 or return code 12 was issued last time the exit was invoked.

UEPRC8
The exit program returned control to CICS on the previous invocation with return code 8.

UEPRC12
The exit program returned control to CICS on the previous invocation with return code 12.

UEPPAD
A 1-byte padding field.

UEPFSPL
Address of the 10-byte function shipping parameter list.

UEPCONST
Address of the 158-byte system entry statistics record (this can be mapped using DSECT DFHA14DS).

UEPMODST
Address of the 84-byte modegroup statistics record for the modegroup specified in the relevant CICS profile. This field applies only to APPC connections for a specific allocate. For LU61, IRC, or non-specific APPC allocates, it contains zero.

The statistics record can be mapped using DSECT DFHA20DS. The modegroup name field (A20MODE) may contain blanks. The record is followed by a fullword of X'FFFFFFFF'.

UEPSTEX
A 6-byte area containing additional current statistics for APPC that are not already in the modegroup statistics record (DFHA20DS). For specific allocates, the numbers refer to the specified modegroup only. For non-specific allocates, they refer to the whole connection—that is, they are the totals of each modegroup.

The 6-byte area contains:

UEPEBND
A halfword binary field containing the number of bound sessions

UEPEWWT
A halfword binary field containing the number of contention winners with tasks

UEPELWT
A halfword binary field containing the number of contention losers with tasks.

UEPEMXQT
A halfword binary field containing the maximum queuing time specified for the connection (MAXQTIME on the CONNECTION resource definition).

UEPMDGST
Address of a set of 84-byte modegroup statistics records—one for each user modegroup for the connection. This field applies only to APPC connections for a non-specific allocate. For LU61, IRC, and APPC specific allocates, it contains zero.

Each statistics record can be mapped using DSECT DFHA20DS. The modegroup name field (A20MODE) may contain blanks. The end of the set of records is indicated by a fullword of X'FFFFFFFF'.

Non-specific allocates data: The following three fields contain data relating to MRO, LU6.1, and non-specific APPC allocates:

UEPSAQTS
A double-word binary field containing the time stamp from the TCT system entry indicating the time the queue of non-specific requests was started.

UEPSACNT
A half-word binary field containing the number of all non-specific allocates processed since the queue was started (see UEPSAQTS for the start time).

UEPSARC8
A half-word binary field containing the number of sessions freed since the queue was last purged as a result of a UEPCAKLL return code to CICS.

Specific allocates data: The following three fields contain data relating to specific modegroup allocates. They are applicable only when UEPMODST is non-zero (that is, it contains the address of the relevant modegroup statistics).

UEPMAQTS
A double-word binary field containing the time stamp from the TCT mode entry indicating the time that the modegroup queue was started for this specific modegroup.

UEPMACNT
A half-word binary field containing the number of all specific allocates for this modegroup processed since the queue was started (see UEPMAQTS for the start time).

UEPMAR12
A half-word binary field containing the number of modegroup sessions freed since the queue was last purged as a result of a UEPCAKLL return code to CICS.

UEPQUELM
A half-word binary field containing the queue limit specified for this connection (QUEUELIMIT on the CONNECTION definition).

Return codes
In the case of an allocate that is about to be queued, use one of the following:

UERCAQUE
Queue the allocate request.

UERCAPUR
Reject the allocate request with SYSIDERR.

UERCAKLL
Reject this allocate request with SYSIDERR. Purge all other queued allocate requests and send an information message to the operator console. CICS also returns SYSIDERR to all application programs waiting on the purged allocate requests.

UERCAKLM
Reject this allocate request for the modegroup and return SYSIDERR. Purge all other queued allocate requests for the

modegroup specified on this allocate request and send an information message to the operator console. Retry the modegroup after an interval.

UERCPURG
Task purged during XPI call.

In the case of a successful allocate following the use of UERCAKLL or UERCAKLM, on a previous invocation of the exit, use one of the following:

UERCNORM
Resume normal operation of the link or modegroup.

UERCAPUR
Reject the allocate request with SYSIDERR.

XPI calls
All can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Designing an XZIQUE global user exit program

The functions of your XZIQUE exit should be designed:

1. To control of the number of tasks (and the amount of associated resource) that are waiting in a queue for a free intersystem session. Waiting tasks can degrade the performance of the local system.
2. To detect poor response from the receiving (remote) system and to notify the operator (or automatic operations program).
3. To cause CICS to issue a message when the link resumes normal operation.

The XZIQUE global user exit parameter list is designed to support these objectives.

Related concepts:

"MRO and APPC session queue management sample exit program (DFH$XZIQ)" on page 29
CICS includes a sample exit program that implements the basic functions provided by the QUEUELIMIT and MAXQTIME parameters on a connection resource definition. The sample program is DFH$XZIQ.

Design considerations: The information passed at XZIQUE is designed to enable your XZIQUE global user exit program to:

- Avoid false diagnosis of problems on the connection by distinguishing poor response times from a complete bottleneck
- Ensure that a link resumes normal operation quickly and without operator intervention once any problem in a remote system is resolved.

Some guidance on the use of IRC/ISC statistics:

CICS adds an entry for unsatisfied allocate requests to the *non-specific (generic) allocate queue*, and the *specific modegroup allocate queue*.

Non-specific (generic) allocate queue
All non-specific allocate requests are queued in this single queue. CICS makes the total number of entries in this queue available in the system entry statistics field A14ESTAQ, to which your global user exit program has access by means of the address of the system entry statistics, which is passed in UEPCONST.

Specific modegroup allocate queues
Specific allocate requests are queued in the appropriate modegroup queue—one queue for each specific modegroup name. CICS makes the total number of entries in all these queues available, as a single total, in the mode entry statistics field A20ESTAQ, to which your global user exit program has access by means of the address of the mode entry statistics, which is passed in UEPMODST.

Sample exit program design:

A sample XZIQUE exit program is provided with CICS Transaction Server for z/OS, Version 5 Release 2 as a base for you to design your own global user exit program.

A sample XZIQUE exit program is provided with CICS Transaction Server for z/OS, Version 5 Release 2 as a base for you to design your own global user exit program. It is called DFH$XZIQ, and is supplied in the CICSTS52.CICS.SDFHSAMP library. The DSECT used by the sample program to map the area addressed by UEPZDATA is called DFHXZIDS, and this is supplied in the CICSTS52.CICS.SDFHMAC library.

As supplied, the sample exit program implements the same basic function as described for the QUEUELIMIT and MAXQTIME parameters on the connection resource definition. If the XZIQUE exit is not enabled, CICS uses these parameters to control the existence and length of the queue of allocate requests. If you enable the exit, the parameters from the connection definition are passed to your XZIQUE global user exit program, which can change the way in which these parameters are used.

The exit program also demonstrates how to control allocate requests for a particular modegroup, based on the same QUEUELIMIT and MAXQTIME parameters.

Overview of the sample exit program: The program uses the exit-specific parameters passed by CICS to determine the state of the connection, and to request the appropriate action, as follows:

1. The connection is operating normally; a queue may exist, but is of short length.

   In this case, the exit program returns with UERCAQUE to indicate that CICS is to **queue the request**.
2. The response from the partner system is slower than the rate of requests demands, and the queue length has grown to the limit specified on the QUEUELIMIT parameter. The partner system is still operating normally, but is overloaded.

   In this case, the exit program returns with UERCAPUR to indicate that CICS is to **purge the request**.

3. The queue has reached the limit specified by the QUEUELIMIT parameter, **and** requests that join the queue are expected to take longer to be satisfied than the time defined by the MAXQTIME parameter. (The estimated time for a request to complete is calculated by dividing the number of successful requests since the queue first formed by the time elapsed since it formed. These statistics are passed to the exit in the parameter list.)

   These criteria are used to determine that the connection is not operating correctly, and that continued queuing of tasks is not helpful. In this case:

   - The exit returns with UERCAKLL requesting CICS to **purge all queued** user requests from the connection. The SYSIDERR condition is returned to the application program.
   - CICS issues message DFHZC2300 to warn that a connection is not performing as expected.

4. The queue has been purged as a result of a previous invocation of the global user exit program, there are still no free sessions, and the request is about to be queued.

   In this case, the exit program returns with UERCAPUR to indicate that CICS is to **purge the request**. This also leaves the UEPRC8 flag set.

5. The queue has been purged as a result of a previous invocation of the global user exit program. A new allocate request has been received and is about to be allocated because a session has become free.

   CICS invokes the exit program to enable it to indicate that normal processing can continue.

   In this case, the exit program returns with UERCNORM to indicate that CICS is to **continue processing normally**. This also causes the UEPRC8 flag to be unset following this invocation, and CICS to issue message DFHZC2301.

The sample program also monitors the length of queues for modegroup-specific allocate requests and controls these—in the same way as the queue for the whole connection—using the QUEUELIMIT parameter and MAXQTIME parameters.

If both UEPRC8 and UEPRC12 are set, UERCNORM is required twice to resume normal operation. The UEPRC8 condition is reset first in this case.

Extensions to the sample program:

The sample exit program does not attempt to control the queue length, or detect poor response for a particular modegroup differently from the whole connection. This kind of enhancement is something you might want to add to your own exit program if your applications request specific modegroups via the allocate command (or via a transaction profile) and you think it would be useful to control the modegroups individually.

You can also use more complex decisions (such as adding time delays to lessen the risk of false diagnosis) to decide when to issue the return codes that purge the queue, and allow queuing to restart.

XISQUE exit for managing IPIC intersystem queues

You can use the XISQUE exit to control queuing on IP interconnectivity (IPIC) connections.

The XISQUE exit controls these requests or commands that are queued on the IPIC connection:

- Distributed program link (DPL) requests for sessions
- Transaction routing requests
- Function shipping requests
- START or CANCEL commands.

Use the XISQUE exit to detect queuing problems (bottlenecks) early.

XISQUE enables allocate requests to be queued or rejected, depending on the length of the queue. It also allows an IPCONN on which there is a bottleneck to be ended and then reestablished.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

"Session queue management sample exit program for IPIC connections (DFH$XISQ)" on page 29
CICS includes a sample program that implements the basic function provided by the QUEUELIMIT and MAXQTIME parameters on a connection resource definition. The sample program is DFH$XISQ.

Exit XISQUE

The XISQUE global user exit is called, if it is enabled, when CICS attempts to acquire a session and no free session is available, or when an allocate request finds a free session after queuing has previously been suppressed. Exit-specific parameters, return codes, and XPI call information are explained.

When called

The XISQUE exit is called under these circumstances:

1. CICS tries to acquire a session on an IPIC connection to a remote system and no free session is available. It is called whether or not you have specified the QUEUELIMIT option on the IPCONN definition and whether or not the limit has been exceeded.

   Requests for IPIC sessions occur when one of the following requests or commands is used across an IPIC connection:

   - A distributed program link (DPL) request
   - A START or CANCEL command
   - A transaction routing request
   - A function shipping file control, transient data, or temporary storage request
2. An IPIC allocate request succeeds in finding a free session, after the queue on the IPIC connection has been purged by a previous call of the exit program. In this case, your exit program can indicate that CICS is to continue processing normally, resuming queuing when necessary.

Exit-specific parameters

UEPISDATA

Address of the 78-byte area. This area is mapped by the DSECT in copybook DFHXIQDS.

Area addressed by UEPISDATA:

UEPREQ
A 2-byte origin-of-request code, which can have the following value:

AL Other kinds of intercommunication (for example, STARTs).

FS Function shipping and distributed program link

TR Transaction routing

UEPIPCNM
The 8-byte name of the IPCONN.

UEPREQTR
The 4-byte identifier of the requesting transaction.

UEPFLAG
A 1-byte flag indicating whether a return code 8 was issued the last time the exit was called.

UEPRC8
The exit program returned control to CICS on the previous call with return code 8.

UEPFSPL
Address of the 10-byte parameter list for the DPL request.

UEPCONST
Address of the 504-byte IPCONN statistics record. This record can be mapped using DSECT DFHISRDS.

UEPEMXQT
A halfword binary field containing the maximum queuing time, MAXQTIME, specified in the IPCONN resource definition.

UEPSAQTS
A double-word binary field containing the time stamp from the installed IPCONN resource definition, indicating the time that the queue of allocate requests was started.

UEPSACNT
A half-word binary field containing the number of allocate requests processed since the queue was started. See UEPSAQTS for the start time.

UEPSARC8
A half-word binary field containing the number of sessions freed since the queue was last purged as the result of an UEPCAKLL return code.

UEPQUELM
A half-word binary field containing the queue limit, QUEUELIMIT, specified in the IPCONN resource definition.

Return codes
In the case of an allocate that is about to be queued, use one of the following return codes:

UERCAQUE
Queue the allocate request.

UERCAKLL
Reject this allocate request with SYSIDERR. Purge all other queued allocate requests and send an information message to the operator console. CICS also returns SYSIDERR to all application programs waiting on the purged allocate requests.

UERCAPUR
Reject the allocate request with SYSIDERR.

UERCPURG
Task purged during XPI call.

In the case of a successful allocate following the use of UERCAKLL on a previous call of the exit, you can use a normal return code (UERCNORM) or use the SYSIDERR return code (UERCAPUR):

UERCNORM
Resume normal operation of the IPCONN.

UERCAPUR
Reject the allocate request with SYSIDERR.

XPI calls
All can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Using an XISQUE global user exit program

When the exit is enabled, your XISQUE global user exit program is able to check on the state of allocate queues for IPCONNs in the local system. The parameter list passed to the exit program on invocation provides data about a specific allocate request and IPCONN.

Using the information passed in the parameter list, your global user exit program can decide (based on queue length, for example) whether CICS is to queue the allocate request. Your program communicates its decision to CICS by setting one of the following return codes:

UERCAQUE
CICS is to queue the allocate request.

The total number of allocate requests currently queued against the connection is provided in field ISR_CURRENT_QUEUED_ALLOCATES of the IPCONN statistics record, which is addressed by the UEPCONST exit-specific parameter. See DSECT DFHISRDS for details.

CICS also passes to the exit program, in field UEPQUELM, the value of the QUEUELIMIT option of the IPCONN resource definition. If the queue limit has not been reached, you can return control to CICS with return code UERCAQUE.

UERCAPUR

CICS is to reject the allocate request, return SYSIDERR to the application program, and leave the existing queue unchanged.

If the number of queued allocate requests has reached the limit set on the QUEUELIMIT option of the IPCONN definition, you can request that CICS rejects the request. However, you should first check whether the state of the link is satisfactory. This means checking that the rate of allocation of sessions is acceptable. Use the time the queue was started, the current time, and the total number of allocates processed since the queue began, to determine the rate at which CICS is processing requests. The relevant fields are: UEPSAQTS and UEPSACNT.

To determine whether CICS is allocating requests for sessions on this IPCONN at an acceptable rate, you can compare the calculated time with either of the following:

1. The value of the MAXQTIME option of the IPCONN resource definition, which is passed in the UEPEMXQT exit-specific parameter.
2. Some other preset time value.

If, using this kind of formula, you find the processing time to be acceptable, return control to CICS with return code UERCAPUR, which purges only this request.

UERCAKLL

Reject this request, purge all other allocate requests queued on this IPCONN, and send an information message to the operator console.

If the queue limit has been reached and the performance of allocate processing is below the acceptable limits defined in your user exit program, you can purge all queued allocate requests by specifying return code UERCAKLL.

UERCAKLL also causes CICS to:

- Return SYSIDERR to all application programs waiting on the purged allocate requests.
- On subsequent calls to your XISQUE exit program, set the UEPFLAG parameter to UEPRC8 to indicate that UERCAKLL was returned previously to purge the queue.

Purging a queue that is causing congestion in the flow of tasks frees task slots that are needed to prevent the system becoming clogged. The more you allow a session queue to grow, the more likely you are to reach the task ceiling set by the MAXT parameter, and then cause a queue of incoming tasks in the local region that cannot be attached.

If a queue has been purged previously (with UERCAKLL) but there are no queued requests currently, check the number of sessions freed since the queue was last purged. This number is in UEPSARC8. If no sessions have been freed on this IPCONNsince the queue was last purged, the problem that caused the purge previously has not been resolved, and this request should be rejected with UERCAPUR.

If the UEPSARC8 parameter shows that sessions are being freed, you should use UERCAQUE to resume queuing of requests. If you return with UERCAQUE in this case, CICS issues an information message to the console to signal that queuing has been resumed.

UERCNORM

CICS is to resume normal processing on the connection, including queuing of requests.

Use UERCNORM when the exit is invoked after a successful allocate following the suppression of queuing.

Statistics fields in DFHISRDS

The following fields in IPCONN statistics can help your XISQUE global user exit program to control allocate queues efficiently:

ISR_XISQUE_ALLOC_REJECTS
Each time an XISQUE global user exit program returns with a request to reject a request, CICS increments this field, which is provided to help you tune the queue limit. Normally, if the number of sessions and the queue limit specified on the IPCONN definition are correctly balanced, and there has been no abnormal congestion on the link, ISR_XISQUE_ALLOC_REJECTS should be zero. If the rejected allocates field is non-zero it indicates that action is probably needed.

ISR_XISQUE_ALLOC_QPURGES
Each time an XISQUE global user exit program returns with a request to purge a queue, CICS increments this field.

For detailed information about the fields in IPCONN statistics, see "IPCONN statistics", in the *CICS Performance Guide*. IPCONN statistics are mapped by DSECT DFHISRDS.

Designing an XISQUE global user exit program

Your XISQUE exit program should be designed to:

1. Control of the number of tasks (and the amount of associated resource) that are waiting in a queue for a free IPIC session. Waiting tasks can degrade the performance of the local system.
2. Detect poor response from the remote system and notify the operator (or automatic operations program).
3. Cause CICS to issue a message when the IPCONN resumes normal operation.

The XISQUE parameter list is designed to support these objectives. The information it contains enables your exit program to:

- Avoid false diagnosis of connection problems by distinguishing poor response times from a complete bottleneck
- Ensure that a link resumes normal operation quickly and without operator intervention after a problem in a remote system is resolved

Using IPCONN statistics

In reaching its decisions about which requests to reject, which to queue, and which queues to purge, your exit program will probably take into account the number of allocate requests currently queued against the connection. All allocate requests for a particular IPCONN are queued in a single queue that is specific to that IPCONN. CICS makes the total number of entries in this queue available in the IPCONN statistics field ISR_CURRENT_QUEUED_ALLOCATES. Your exit program can access this field by means of the address of the IPCONN statistics, which is passed in the UEPCONST exit-specific parameter.

The sample exit program, DFH$XISQ

CICS provides a sample XISQUE exit program, DFH$XISQ, that you can use as the basis for your own program. It is supplied in the CICSTS52.CICS.SDFHSAMP

library. The DSECT used by the sample program to map the area addressed by UEPISDATA is called DFHXIQDS, and is supplied in the CICSTS52.CICS.SDFHMAC library.

As supplied, the sample exit program implements the same basic function as described for the QUEUELIMIT and MAXQTIME parameters of the IPCONN resource definition. If the XISQUE exit is not enabled, CICS uses these parameters to control the existence and length of the queue of allocate requests. If you enable the exit, the parameters from the IPCONN definition are passed to the XISQUE exit program, which can change the way in which these parameters are used.

Overview of the sample exit program

The program uses the exit-specific parameters passed by CICS to determine the state of the IPCONN, and to request the appropriate action, as follows:

1. The IPCONN is operating normally; a queue may exist, but is of short length.

   In this case, the exit program returns with UERCAQUE to indicate that CICS is to queue the request.

2. The response from the partner system is slower than the rate of requests, and the queue length has grown to the limit specified on the QUEUELIMIT parameter. The partner system is still operating normally, but is overloaded.

   In this case, the exit program returns with UERCAPUR to indicate that CICS is to purge the request.

3. The queue has reached the limit specified by the QUEUELIMIT parameter, and requests that join the queue are expected to take longer to be satisfied than the time defined by the MAXQTIME parameter. (The estimated time for a request to complete is calculated by dividing the number of successful requests since the queue first formed by the time elapsed since it formed. These statistics are passed to the exit in the parameter list.)

   These criteria are used to determine that the connection is not operating correctly, and that continued queuing of tasks is not helpful. In this case:

   - The exit returns with UERCAKLL, requesting CICS to purge all queued user requests from the connection. The SYSIDERR condition is returned to the application program.
   - CICS issues message DFHZCaaaa to warn that a connection is not performing as expected.

4. The queue has been purged as a result of a previous invocation of the global user exit program, there are still no free sessions, and the request is about to be queued.

   In this case, the exit program returns with UERCAPUR to indicate that CICS is to purge the request. This also leaves the UEPRC8 flag set.

5. The queue has been purged as a result of a previous invocation of the global user exit program. A new allocate request has been received and is about to be allocated because a session has become free.

   In this case, the exit program returns with UERCNORM to indicate that CICS is to continue processing normally. This also causes the UEPRC8 flag to be unset and CICS to issue message DFHZCbbbb.

Extending the sample program

The sample exit program does not attempt to control the queue length. This kind of enhancement is something you might want to add to your own exit program.

You can also use more complex decisions (such as adding time delays to lessen the risk of false diagnosis) to decide when to issue the return codes that purge the queue, and allow queuing to restart.

XRF request-processing program exit XXRSTAT

This exit is invoked if a z/OS Communications Server failure or a z/OS Communications Server predatory takeover occurs.

XXRSTAT enables you to decide whether to terminate CICS when either of the following occurs:

- CICS is notified of a z/OS Communications Server failure by the TPEND exit.
- A predatory takeover has occurred. Predatory takeover can occur if you are using Release 3.4.0 or above, and a z/OS Communications Server application with the same APPLID as that of the executing CICS system assumes control of all the sessions of the executing CICS system.

  XXRSTAT gives you the choice of allowing the system which has suffered the takeover to continue or to terminate.

  To avoid potential integrity exposures, CICS default action after a predatory takeover is to terminate without a dump. If you want CICS to terminate with a dump, your exit program should return UERCABDU. CICS terminates with the abend code specified by your exit program.

  If you want CICS to continue after a predatory takeover, your exit program must return UERCCOIG. Message DFHZC0101 is issued and CICS continues processing without z/OS Communications Server support. The predatory application assumes control of all z/OS Communications Server sessions.

  Note: Allowing CICS to continue after a predatory takeover could cause integrity problems and is not recommended. Use RACF to protect your CICS APPLIDs.

Related concepts:

"Writing global user exit programs" on page 3
You must write global user exit programs in assembler language and they must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.

Exit XXRSTAT

This exit is invoked if a z/OS Communications Server failure or predatory takeover occurs.

When invoked

After either of the following:

- CICS is notified of a z/OS Communications Server failure by the TPEND exit.
- A predatory takeover.

Exit-specific parameters

UEPERRA

Address of parameter list containing:

UEPGAPLD

Address of the 8-byte generic applid

UEPSAPLD

Address of the 8-byte specific applid

UEPDOMID
Address of the 4-byte domain ID

UEPERRID
Address of the 4-byte error ID.

Note:

1. No DSECT is provided for this parameter list. You need to code your own DSECT to access the named fields.
2. When z/OS Communications Server has failed, the domain ID is 'ZC ' (uppercase Z, uppercase C, and two blanks), and the error ID is the character string '3443'.

Return codes

UERCNORM
Take the system action. The system action depends on the reason why the exit was invoked:

- For XRF, in the event of a z/OS Communications Server failure: CICS continues processing as if the exit program had not been invoked.
- For z/OS Communications Server persistent sessions, in the event of a predatory takeover: CICS abends without a dump.

UERCCOIG
Ignore.

UERCABNO
Abend CICS without a dump.

UERCABDU
Abend CICS with a dump.

UERCPURG
Task purged during XPI call.

XPI calls
All can be used.

Related concepts:

"Overview of the XPI" on page 379
The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

"Making an XPI call" on page 382
An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

Chapter 2. Task-related user exit programs

Use a *task-related user exit* (TRUE) to write your own program to access a resource, such as a database, that is not otherwise available to your CICS system.

Introduction to the task-related user exit mechanism (the adapter)

You can use a task-related user exit (TRUE) to write your own program to access a resource, such as a database, that would not otherwise be available to your CICS system.

Such a resource is known as a non-CICS resource. The exit is said to be task-related because it becomes part of the task that invoked it and because, unlike a global user exit, it is not associated with an exit point. You do not have to use any of the task-related user exits, but you can use them to extend and customize the function of your CICS system according to your own requirements.

The most common use of a task-related user exit is to communicate with a resource manager external to CICS; for example, a file or database manager. The CICS interface modules that handle the communication between the task-related user exit and the resource manager are referred to as the resource manager interface (RMI) or the task-related user exit interface.

The task-related user exit mechanism is known as an *adapter* because it provides the connection between an application program that must access a non-CICS resource, and the manager of that resource. Figure 10 on page 338 illustrates the adapter concept.

The adapter is made up of three or more locally written programs. These are a "stub" program, a task-related user exit program, and one or more administration routines or programs.

The *stub program* intercepts a request (for example, to access data held on an external database manager) issued by the calling application program. The stub can be used to resolve a locally defined high-level language command into a task-related user exit macro call, DFHRMCAL, which then causes CICS to pass control to the task-related user exit program.

The *task-related user exit program* translates commands for accessing a non-CICS resource into a form acceptable to the resource manager. The program must be written in assembler language, and can reside below the 16 MB line, or above 16 MB but below 2 GB. For more information about addressing and residency modes, see "Addressing-mode implications" on page 361. The program must not alter the contents of any access registers. It is executed in response to a specific application program request, for example, to read data from an external database. In this instance, it can be passed application data, such as a search argument for a required record. Responses from the resource manager are passed back to the calling program by the task-related user exit program.

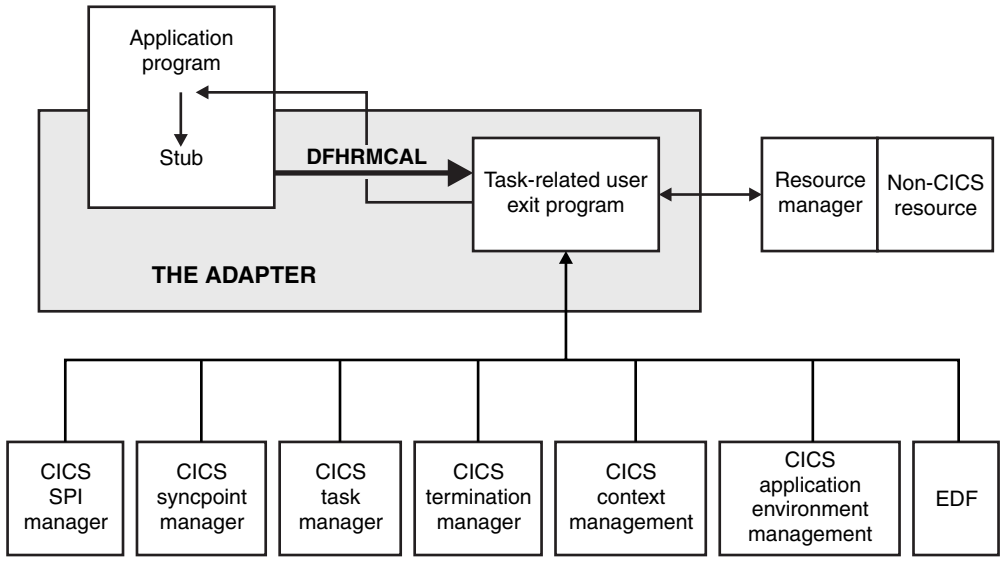


Figure 10. The adapter concept

The task-related user exit program is provided with a parameter list (DFHUEPAR) by the CICS management module that handles task-related user exits. This parameter list gives the task-related user exit access to information such as the addresses and sizes of its own work areas.

The task-related user exit program can be invoked by any of the following:
- An application program
- CICS SPI manager
- CICS sync point manager
- CICS task manager
- CICS termination manager
- CICS context management
- CICS application environment management
- The Execution Diagnostic Facility (EDF)

The parameter list serves to distinguish between these various callers, and gives access to a register save area containing the registers of the caller.

AMODE 64 task-related user exits are not supported.

The *administration routines* contain the EXEC CICS ENABLE and DISABLE commands that you use to install and withdraw the task-related user exit program. The administration routines might also contain commands to retrieve information about one of the work areas of the exit program (the EXEC CICS EXTRACT EXIT command), and to resolve any inconsistency between CICS and a non-CICS resource manager after a system failure (the EXEC CICS RESYNC command). For detailed programming information, see ENABLE PROGRAM command in Reference > System programming, DISABLE PROGRAM command in Reference > System programming, EXTRACT EXIT in Reference > System programming, and RESYNC ENTRYNAME in Reference > System programming.

The stub program

The stub program shields application programmers from the mechanics of non-CICS resource managers. It is written in assembler language. After assembly, the stub is link-edited to each application program that wants to use it.

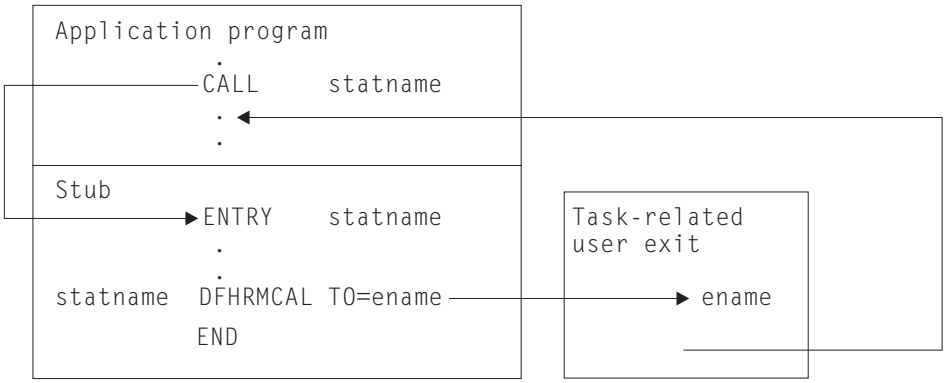


Figure 11. The stub concept

statname
A label that can be referenced externally. Statname must conform to the requirements of an assembler language ENTRY statement, and typically resolves a V-type address constant, or the target of a high-level language CALL. A single stub can contain several such labels.

ename
The entry name (specified on the **EXEC CICS ENABLE** command) of the task-related user exit program that you want to handle resource manager requests.

You can define high-level language commands for your programmers to use when they want to access a non-CICS resource. You must use a translator to convert a locally defined high-level language command into a conventional CALL to the required entry point of the stub program. Alternatively, the application program can issue a CALL naming the stub entry point, as shown in Figure 11. For example, to read a record from a non-CICS resource, an application program can use the following COBOL statement:

```
CALL 'XYZ' USING PARM1 PARM2...
```

XYZ is an entry point (the statname) in your stub program. The stub converts the command into a macro call (DFHRMCAL) to the task-related user exit program, specified in the TO= operand. Return from the task-related user exit program is to the calling application program, not to the stub program.

The application can use a parameter to determine whether the resource manager was called. For example, if the application sets a parameter to zero and the resource manager sets it to nonzero, the parameter value on return indicates whether the resource manager was invoked.

Notes:

- You can use only the TO, RTNABND, and SUPPEDF operands of the DFHRMCAL macro. Any other operands are for CICS internal use only.

- The DFHRMCAL macro cannot be invoked by an AMODE(64) application program.

Returning control to the application program

If you specify RTNABND=YES in the DFHRMCAL macro, control returns to the application program when the task-related user exit is unavailable, for example, because it is not enabled or started.

Note that for assembler language application programs, a negative value in register 15 signals to the application program that control has returned because the exit is unavailable. The task-related user exit program can use positive values (including zero) in register 15 to pass resource manager response codes to the application program.

If you do not specify RTNABND=YES and the task-related user exit is unavailable, the application program terminates abnormally with the abend code 'AEY9'.

Task-related user exits and EDF

When a task-related user exit (TRUE) is invoked for a call to a non-CICS resource manager from an application that is being monitored by EDF, the default action of EDF is to display the parameters that are addressed by the parameter list passed by the DFHRMCAL macro. By specifying FORMATEDF on the EXEC CICS ENABLE command that enables the TRUE the parameter list can be transformed into a more meaningful display.

The TRUE is invoked several times, before and after the invocation to satisfy the call to the resource manager, to format the data to be displayed by EDF and to deal with any changes made by the user to the data on the EDF screen.

For more information about how to format screens for EDF, see "CICS EDF build parameters" on page 356 and "Using EDF with your task-related user exit program" on page 374.

If a task-related user exit program contains EXEC CICS commands, EDF can be useful in debugging the TRUE itself. If you want EDF to display commands from the TRUE, you must specify the EDF option when the TRUE program is translated. The standard EDF screens for the CICS commands are then displayed between the "About to Execute" and "Command Execution Complete" screens for the call to the resource manager. However, as EDF is primarily an application debugging tool and the CICS commands within the TRUE would not generally be of interest to the application programmer, the TRUE program is normally translated with the "NOEDF" option; in this case, screens for CICS commands within the TRUE are suppressed.

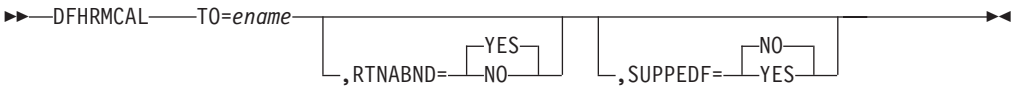
Note: If you specify SUPPEDF=YES on the DFHRMCAL macro, the "About to Execute" and "Command Execution Complete" screens relating to the invocation of the TRUE by DFHRMCAL are suppressed; in other words, DFHRMCAL becomes "invisible" to EDF. Specifying SUPPEDF=YES has no effect in determining whether EDF displays EXEC CICS commands within the TRUE but it does suppress the display of parameters passed to the TRUE.

DFHRMCAL macro

The DFHRMCAL macro passes control to a task-related user exit program (TRUE) that you want to handle resource manager requests.

Syntax

DFHRMCAL



Other parameters for this macro are for CICS internal use only.

This macro cannot be invoked by an AMODE(64) application program.

Parameters

TO=*ename*

ename is the entry name of the task-related user exit program (TRUE) that you want to handle resource manager requests.

RTNABND={YES|NO}

Whether control returns to the application program when the task-related user exit program is unavailable, for example, because it is not enabled or started. Valid values are as follows:

YES

Control returns to the application program when the task-related user exit program is unavailable. This value is the default.

NO Control does not return to the application program when the task-related user exit program is unavailable.

For assembler language application programs, a negative value in register 15 signals to the application program that control has returned because the exit is unavailable. The task-related user exit program can use positive values (including zero) in register 15 to pass resource manager response codes to the application program.

If RTNABND=NO is specified and the task-related user exit program is unavailable, the application program terminates abnormally with the abend code AEY9.

SUPPEDF={YES|NO}

Whether EDF screens that relate to the invocation of the TRUE by DFHRMCAL are suppressed. When a TRUE is invoked for a call to a non-CICS resource manager from an application that is being monitored by EDF, EDF can display "About to Execute" and "Command Execution Complete" screens that show parameters that are addressed by the parameter list that the DFHRMCAL macro passes. Valid values are as follows:

YES

Suppress EDF screens that relate to the invocation of the TRUE by DFHRMCAL.

NO Display EDF screens that relate to the invocation of the TRUE by DFHRMCAL. This value is the default.

Writing a task-related user exit program

The main function of the task-related user exit program is to translate the calling program's parameters into a form acceptable to your non-CICS resource manager, and then to pass control to the resource manager.

The calling program's parameters are described in “Caller parameter lists” on page 350.

This section describes the user exit parameter lists, the schedule flag word, which is used by the exit program to register its need to be invoked by CICS management services, and register-handling in the task-related user exit program. This section also discusses the use of the CICS syncpoint manager and the CICS task manager. It also discusses some factors that you should consider if you plan to use TCBs provided by the CICS open transaction environment (OTE).

Obligations of OPENAPI task-related user exits (TRUEs)

Open API TRUEs (that is, exits enabled with the OPENAPI option) are invoked on an open L8 mode TCB instead of the main CICS QR TCB. If a TRUE wants to invoke an external resource manager without the benefit of the open API facility, it must manage its own set of subtask TCBs.

Quasirent TRUEs are invoked on QR TCB and create their own subtask TCBs. Typically, the subtask TCB is posted to access the external resource manager, while the CICS task running on the QR TCB in the TRUE is put into a CICS dispatcher wait until the subtask completes its work. The CICS dispatcher wait allows the CICS dispatcher to dispatch another CICS task on the QR TCB in the meantime. The reason for this architecture is that external resource managers cannot be invoked directly by a caller on the QR TCB, because any operating system wait issued by the external resource manager would halt the QR TCB, and the whole of CICS.

An open API TRUE is invoked on an L8 mode TCB, which is dedicated for use by the calling CICS task, and is separate from the CICS QR TCB. Therefore, a TRUE can invoke its external resource manager on the allocated L8 TCB and avoid the need to manage its own set of subtask TCBs. An operating system wait issued by an external resource manager halts the L8 TCB, but CICS continues processing on the QR TCB, and on other open TCBs.

An open API TRUE, although freed from the constraints imposed by the QR TCB, and from having to create subtask TCBs, nevertheless does have obligations both to the CICS system as a whole and to future users of the L8 TCB it is using. An L8 TCB is dedicated for use by the CICS task to which it is allocated, but when the CICS task has completed, the L8 TCB is returned to the dispatcher-managed pool of L8 mode TCBs, provided it is still in a clean state. An unclean TCB in this context means that the task using the L8 mode TCB suffered an unhandled abend in an open API TRUE, and not that the TRUE has broken the threadsafe restrictions, which CICS would not detect. The L8 TCB is not dedicated for use by a particular open API TRUE, but is used by all open API TRUEs and OPENAPI programs invoked by the CICS task to which the L8 mode TCB is allocated. Also, if an application program invoking an open API TRUE is coded to threadsafe standards, and defined to CICS as threadsafe, it continues to run on the L8 mode TCB on return from the TRUE. When considering whether to use CICS OTE support rather than manage your own subtask TCBs, consider the restrictions

listed in “Threadsafe restrictions.” You can then decide whether to work within these restrictions or to use your own subtask TCB, which is dedicated for your own TRUE's use.

When a TRUE is enabled REQUIRED and OPENAPI, it is treated the same as if it were enabled THREADSAFE and OPENAPI. For consistency, an **INQUIRE EXITPROGRAM** for either combination will always return THREADSAFE, OPENAPI. For a task-related user exit enabled REQUIRED and CICSAPI, **INQUIRE EXITPROGRAM** will return REQUIRED, CICSAPI.

Threadsafe restrictions

An open API TRUE must not treat the executing open TCB environment in such a way that it causes problems for:

- Other open API TRUEs called by the same task
- OPENAPI programs called by the same task
- Application program logic that could run on the open TCB
- Future tasks that might use the open TCB
- CICS management code.

In particular:

- When invoking CICS services, or when returning to CICS, an open API TRUE must ensure it restores the MVS programming environment as it was on entry to the TRUE. This includes cross-memory mode, ASC mode, request block (RB) level, linkage stack level, TCB dispatching priority, in addition to cancelling any ESTAEs added.
- At CICS task termination, an open API TRUE must ensure it leaves the open TCB in a state suitable to be reused by another CICS transaction. In particular, it must ensure that all non-CICS resources acquired specifically on behalf of the terminating task are freed. Such resources might include:
  - Dynamically allocated data sets
  - Open ACBs or DCBs
  - STIMERM requests
  - MVS managed storage
  - ENQ requests
  - Attached subtasks
  - Loaded modules
  - Owned data spaces
  - Added access list entries
  - Name/token pairs
  - Fixed pages
  - Security settings (TCBSENV must be set to zero)
- An open API TRUE must not use the following MVS system services that will affect overall CICS operation:
  - CHKPT
  - ESPIE
  - QEDIT
  - SPIE
  - STIMER
  - TTIMER
  - XCTL / XCTLX
  - Any TSO/E services.

- An open API TRUE must not invoke under the L8 mode TCB a Language Environment program that is using MVS Language Environment services, because L8 mode TCBs are initialized for Language Environment using CICS services.

Calling an OPENAPI task-related user exit

About this task

If a task-related user exit is enabled with the OPENAPI option, CICS uses the following rules, based on the type of call, to determine the TCB on which it should invoke the TRUE—on the QR TCB, on the caller's TCB, or an L8 mode TCB:

Application program call (API)—UERTAPPL
For this call, CICS always invokes the TRUE on an L8 mode TCB

CICS syncpoint manager call—UERTSYNC
For this call, CICS always invokes the TRUE on an L8 mode TCB

CICS task manager call—UERTTASK
For this call, the TCB on which CICS invokes the TRUE is further determined by the type of task manager call:

UERTSOTR—Start of task
For this call the open API option is ignored for performance reasons, and CICS always invokes the TRUE on the QR TCB.

UERTEOTR —End of task
For this call, CICS always invokes the TRUE on an L8 mode TCB.

EDF call—UERTFEDF
For this call, CICS always invokes the TRUE on an L8 mode TCB

CICS SPI call—UERTSPI
For this call, CICS always invokes the TRUE as a threadsafe TRUE on the TCB on which the task is currently running at the time of the call.

The SPI function, which is to satisfy EXEC CICS INQUIRE EXITPROGRAM commands on which the CONNECTST or QUALIFIER option are specified, is simple and does not require invocation on a specific TCB.

CICS termination call—UERTCTER
For this call the open API option is ignored and CICS always invokes the TRUE on the QR TCB.

Note: Even for call types that are invoked on an L8 TCB, it is possible that the L8 TCB could suffer an aynschronous abend and therefore not be available for subsequent use, with the following result:

- If CICS is unable to switch to the L8 TCB for an API call to a TRUE, CICS abends the transaction.
- If CICS is unable to switch to the L8 TCB for a syncpoint or end of task call, CICS invokes the TRUE on the QR TCB instead.

The TCB mode on which the task-related user exit is being called is provided in the second and third bytes of a three-byte filed addressed by the UEPTIND parameter. See User exit parameter lists for details.

User exit parameter lists

When a task-related user exit program is invoked, the CICS management module that handles task-related user exits provides the exit program with a parameter list. The address of this parameter list is passed in register 1.

The list contains the following information:

- The identity of the caller
- Addresses and sizes of any work areas that are available to the task-related user exit program
- The address of the register save area of the caller
- The address of an EXEC interface block (EIB) that is for use by the task-related user exit program during this invocation
- The address of the identifier of the current unit of recovery
- The address of the schedule flag word
- The address of the kernel stack entry
- The address of the APPC unit of work (UOW) identifier
- The address of the user security block flag
- The address of the user security block
- The address of the resource manager qualifier name
- The address of the resource manager's single-update and read-only indicator byte
- The address of the caller's AMODE indicator byte
- The address of the application's DATALOC and TASKDATAKEY indicator byte
- The address of the performance block token
- The address of a trace flag.

To enable your exit program to access this parameter list, you must include in it the macro as follows:

```
DFHUEXIT TYPE=RM
```

The DFHUEXIT TYPE=RM macro causes the assembler to create the storage definitions (DSECTs) DFHUEPAR, DFHUERTR, and DFHUECON. If you want your task-related user exit to be able to format screens for EDF, you must include in it the macro as follows:

```
DFHUEXIT TYPE=RM,DSECT=EDF
```

This causes the assembler to create the UEPEDFRM DSECT, which is described in "CICS EDF build parameters" on page 356. All of the user exit parameter lists are summarized in "Summary of the task-related user exit parameter lists" on page 357.

The following information describes the format and the purpose of these definitions.

DFHUEPAR

DFHUEPAR gives you the following symbolic names for address parameters:

UEPEXN
: Address of the function definition, which tells the task-related user exit program why it is being called. See "DFHUERTR (the function definition)" on page 349 for more details.

UEPGAA
: Address of the global work area requested in the EXEC CICS ENABLE command. The global work area is described in "Work areas" on page 364. CICS initializes this work area to X'00' when the task-related user exit program is enabled.

UEPGAL
Address of a halfword containing the length (binary value) of the global work area.

UEPTCA
This field is retained for historical reasons. Do not reference it in your exit program.

UEPCSA
This field is retained for historical reasons. Do not reference it in your exit program.

UEPHMSA
Address of the register save area (RSA) of the caller. It is an 18-word save area, with the contents of registers 14 through 12 stored in the fourth and subsequent words. Its fifth word, representing register 15 of the calling program, is cleared by CICS before the task-related user exit program is invoked. The fifth word is cleared so that it can be used to convey response codes from the resource manager to the calling program. For this reason, you cannot use register 15 to send data to the task-related user exit program. The seventh word of the save area contains register 1 of the caller. Register 1 addresses the parameter list of the caller. For a summary, see "Summary of the task-related user exit parameter lists" on page 357. When the caller is an application program, the contents of register 1 are determined by the linkage conventions of the language interface of the adapter.

UEPTAA
Address of the local work area requested in the EXEC CICS ENABLE command. The local work area is described in "Work areas" on page 364. CICS initializes the work area to X'00' throughout on first acquiring the area; that is, when the task first invokes the task-related user exit program.

UEPTAL
Address of a halfword containing the binary length of the local work area.

UEPEIB
Address of the EXEC interface block (EIB) created by CICS for the task-related user exit program. The EIB exists only for the duration of the call and it allows the task-related user exit program to request CICS services through the command-level interface. This EIB is not the same EIB that is available to the calling program. You therefore cannot access the environment of the calling program other than by UEPHMSA, which provides the address of the register save area (RSA) of the calling program.

UEPURID
Address of CICS unit of recovery identifier. This field contains the 8-byte date and time value that is generated by an STCK instruction, and it identifies the current unit of work.

UEPFLAGS
Address of the schedule flag word. This is a fullword that the task-related user exit program uses to register its need for the services of CICS management programs. For more information, see "The schedule flag word" on page 358.

UEPRMSTK
Address of the kernel stack entry.

UEPUOWDS
Address of the APPC unit of work (UOW) identifier.

UEPSECFLG
Address of the user security flag. The user security flag is a 1-byte field that can take the following values:

UEPNOSEC (X'80')
Security is not active for this CICS system.

UEPSEC (X'20')
Security is active for this CICS system. Only in this case is the address of the "user security block" set.

UEPSECBLK
Address of a fullword that addresses the user security block, that is, the ACEE.

UEPRMQUA
Address of an 8-byte field into which the task-related user exit can move the qualifier name of the resource manager on each API request. This practice is useful where the same exit program is used to connect to more than one instance of a resource manager. The qualifier identifies the instance of the resource manager to which the exit is currently connected.

Where different resource manager qualifiers are returned on the responses to various API requests within a UOW, it is the resource manager qualifier returned on the final API request immediately before a prepare or backout invocation that is used when recording any indoubt information.

UEPCALAM
Address of the AMODE indication byte of the caller.

X'80' Indicates that the original caller was in AMODE 31. If the top bit is not set and bit 31 is 0, the caller was in AMODE 24.

UEPSYNCA
Address of the single-update and read-only indication byte. This field contains flags that your exit program can set to indicate that the resource manager "understands" the single-update protocol, and to record the status of the current unit of work (UOW). See "Increasing efficiency: single-update and read-only protocols" on page 366.

UEPSUPDR (X'80')
The resource manager understands the single-update protocol. That is, your exit program can instruct the resource manager to perform a single-phase commit, in appropriate circumstances.

UEPREADO (X'40')
The resource manager understands the read-only protocol, and has been in read-only mode for this unit of work so far. (If this flag is not set, it means either that the UOW contains updates for this resource manager, or that the UOW may be read-only but the resource manager does not understand the read-only protocol.)

UEPTIND
Address of a 3-byte field containing indicators.

The first indicator byte can take one of three symbolic values, UEPTANY, UEPTCICS, and UEPTUTCB, which you can test to determine:

- Whether data locations can be above or below 16 MB.
- Whether the application's storage is in CICS-key or user-key storage.
- Whether the TRUE has been called by an unexpected TCB.

UEPTANY (X'80')
The application can accept addresses above 16 MB. If the symbolic value is not UEPTANY, the application must be returned an address below 16 MB.

UEPTCICS (X'40')
The working storage and task lifetime storage for the application are in CICS-key storage (TASKDATAKEY=CICS). If the symbolic value is not UEPTCICS, the working storage for the application and the lifetime storage for the task are in user-key storage (TASKDATAKEY=USER).

UEPTUTCB (X'20')
Indicates an unexpected TCB. Set on a sync point or end-of-task call only, this value indicates a failure to switch to the TCB expected by the task-related user exit. In these two cases, the task-related user exit is called on the QR TCB with the UEPTUTCB bit set. For all other calls, CICS abends the transaction without invoking the task-related user exit.

The second and third bytes contain a value indicating the TCB mode of its caller. This value is represented in DFHUEPAR as both a two-character code and a symbolic value, as follows:

Table 14. TCB indicators in DFHUEPAR

Symbolic value	2-byte code	Description
UEPTQR	QR	The quasi-reentrant mode TCB
UEPTRO	RO	The resource-owning mode TCB
UEPTCO	CO	The concurrent mode TCB
UEPTSZ	SZ	The FEPI mode TCB
UEPTRP	RP	The ONC/RPC mode TCB
UEPTFO	FO	The file-owning mode TCB
UEPTSL	SL	The sockets listener mode TCB
UEPTSO	SO	The sockets mode TCB
UEPTS8	S8	The secure sockets layer mode TCB
UEPTD2	D2	The CICS DB2 housekeeping mode TCB
UEPTL8	L8	An L8 open TCB, used for OPENAPI TRUEs, or OPENAPI programs that are in CICS key
UEPTL9	L9	An L9 open TCB, used for OPENAPI programs that are in user key
UEPTEP	EP	Event processing TCB
UEPTTP	TP	The TP open TCB, used to own the Language Environment enclave and THRD TCB pool for a JVM server.
UEPTT8	T8	A T8 TCB, used by a JVM server to process multithreaded processing.
UEPTX8	X8	An X8 open TCB, used for C and C++ programs, compiled with the XPLINK option, that are in CICS key
UEPTX9	X9	An X9 open TCB, used for C and C++ programs, compiled with the XPLINK option, that are in user key

UEPPBTOK
Address of a 4-byte field containing the z/OS Workload Manager (WLM) Performance Block Token. An exit program can use this token to:

- Access information (such as the service class token, SERVCLS) in the WLM Performance Block. To do so, it must use the WLM EXTRACT macro, IWMMEXTR, passing the Performance Block Token as the MONTKN input parameter. For more information about the IWMMEXTR macro, see *z/OS MVS Programming: Workload Management Services*.
- Relate its resource manager's performance blocks for the work request with the original CICS performance block. For example, DBCTL and DB2 need to correlate the work they do on behalf of CICS with the originating CICS task, so that the z/OS Workload Manager can measure the performance of the whole CICS task. To correlate the work it must use the WLM IWMMRELA macro.

An exit program must make no assumptions about the contents of the Performance Block and *must not attempt to modify it*: if it does so, the results are unpredictable.

UEPTRCE
Address of a 1-byte trace flag indicating whether RMI tracing (the RI trace component) is active.

UEPTRLV1 (X'80')
RMI level 1 trace is active.

UEPTRLV2 (X'40')
RMI level 2 trace is active.

When the task-related user exit has addressed this field, it might, for example, issue an EXEC CICS SET TRACETYPE command to reset the level of RMI tracing.

DFHUERTR (the function definition)

The function definition identifies the caller of the task-related user exit program. The DSECT contains two symbolic definitions (fields).

UERTFGP
A single byte that is set to X'00'. The zero setting shows that this is a task-related user exit invocation and that the parameter list therefore includes the fields UEPTAA, UEPTAL, UEPEIB, UEPURID, and UEPFLAGS.

UERTFID
A single-byte identifier that shows whether this call has been made by an application program, the CICS SPI manager, the CICS syncpoint manager, the CICS task manager, the CICS termination manager, CICS context management, CICS application environment management, or by EDF. It can have one of the following seven settings:

UERTSPI
(X'01') CICS SPI call.

UERTAPPL
(X'02') Application program call.

UERTSYNC
(X'04') CICS syncpoint manager call.

UERTTASK
(X'08') CICS task manager call.

UERTCTER
(X'0A') CICS termination call.

UERTFEDF
(X'0C') EDF call.

UERTSWAE
(X'OD') Switch application environment call.

UERTFCON
(X'0E') CICS context management call.

It is important to know which type of program has made the call because it affects how the calling program's parameter list is interpreted by the task-related user exit program.

Caller parameter lists

In addition to the DSECTs DFHUEPAR and DFHUERTR, the inclusion of DFHUEXIT TYPE=RM in the task-related user exit program provides some field definitions that are specific to the caller of the task-related user exit. The calling program's parameter list is normally addressed by R1 in the calling program's RSA. This RSA is addressed by field UEPHMSA of DFHUEPAR. These parameters are described in the following subtopics.

Application program parameters:

If the caller is an application program, the format and addressing of its parameter list are decided locally.

CICS SPI parameters:

If you enable your task-related exit program with the SPI option of the **EXEC CICS ENABLE PROGRAM** command, the exit program can be started when you use the **EXEC CICS INQUIRE EXITPROGRAM** command or on which the CONNECTST or QUALIFIER option is specified.

Use the **INQUIRE EXITPROGRAM** command to query whether the exit program is connected to its resource manager, and its entryname qualifier. For information about the **INQUIRE EXITPROGRAM** command, see INQUIRE EXITPROGRAM in Reference > System programming.

The CICS SPI parameter list contains two entries:

Parameter 1
The address of a 1- byte output field, which your task-related exit program uses to indicate whether it is connected to its external resource manager. The equated return code values are as follows:

UERTCONN
(X'80') The exit is connected to its resource manager.

UERTNCONN
(X'40') The exit is not connected to its resource manager.

Parameter 2
The address of an 8- character output field, in which your task-related exit

program returns the qualifier of the external resource manager, if known. See the UEPRMQUA parameter in “DFHUEPAR” on page 345 for more information about qualifier names.

CICS syncpoint manager parameters:

The CICS syncpoint manager's parameter list contains ten entries, although on most invocations only parameters 1 and 10 contain values. The operation bytes pointed to by parameters 1 and 10 contain flags which, when combined, form an operation code that tells the TRUE why it has been invoked.

Parameters 2 through 9 contain values only when the syncpoint manager makes a “Commit Unconditionally” or “Backout” call to the TRUE, for resynchronization purposes after a session or system failure. These extra parameters point to fields that identify the task, the transaction that started the task, the terminal from which it was initiated, the identity of the terminal operator, the date and time of the failing syncpoint, and (if there are no further units of recovery associated with the task) the next transaction code. Typically, you would use these values to create meaningful messages for resource recovery. They are presented explicitly because, after a system failure, the task driving the exit is not the task that originally scheduled the recoverable work. These additional parameters describe the **original** task's environment and are accessed directly.

The full parameter list is as follows:

Parameter 1
The address of operation byte 1, which contains the following flags:

UERTPREP
(X'80') Prepare to commit (that is, perform the first phase of a two-phase commit).

UERTCOMM
(X'40') Commit unconditionally (perform the second phase of a two-phase commit).

UERTBACK
(X'20') Backout.

UERTDGCS
(X'10') Unit of recovery has been lost because of an initial start of CICS.

UERTDGNK
(X'08') Resource manager should not be in doubt about this unit of recovery.

UERTWAIT
(X'04') Resource manager must wait for the outcome of this unit of recovery. This value is set at phase two of a two-phase commit, if CICS is indoubt about the outcome of a UOW. It occurs only if the task-related user exit is enabled with the INDOUBTWAIT option (see “Enabling for specific invocation-types” on page 376).

UERTRSYN
(X'02') This syncpoint request was generated as the result of an EXEC CICS RESYNC command.

UERTLAST
(X'01') There are no further units of recovery associated with this task. Note that when this bit is **not** set, there may or may not be further units of recovery. For this reason, it is not recommended that you rely

on this bit to signal end-of-task. You should instead schedule the CICS task manager to drive you at end-of-task by setting the task manager bit in the schedule flag word. If you do use UERTLAST to signal end-of-task, and if at that stage you can complete your clean-up process, you can set the task manager bit off in the schedule flag word when the clean-up process is finished, to avoid an unnecessary invocation by the CICS task manager.

The only **valid bit combinations** are those produced by combining one of UERTPREP, UERTCOMM, UERTBACK, UERTDGCS, and UERTDGNK with either UERTLAST or UERTRSYN, or both; or by combining UERTWAIT and UERTLAST.

Your exit program should examine the flags set both in this byte and in operation byte 2 (see parameter 10), to determine what action is expected of it.

Parameter 2
If not zero, the address of a 4-byte, packed-decimal field identifying the original task. But note that, on many invocations of the exit program, parameters 2 through 9 do not contain values. See note 1.

Parameter 3
Address of a 4-character field identifying the transaction that started the original task. See note 1.

Parameter 4
Address of a 4-character field identifying the terminal from which the original task was initiated. See note 1.

Parameter 5
Address of a 4-character field containing the identity of the terminal operator (OPID) who initiated the original task. See note 1.

Parameter 6
Address of a 4-byte, packed-decimal field containing the date of the failing syncpoint, in the format 0Cyyddds, where:

- **C** is a century indicator. (0=1900, 1=2000, 2=2100, and so on.)
- **yy**=years.
- **ddd**=days.
- **s** is the sign.

See note 1.

Parameter 7
Address of a 4-byte, packed-decimal field containing the time of the failing syncpoint, in the format 0hhmmss+. See note 1.

Parameter 8
Address of an 8-byte field containing the resource manager qualifier. See note 1.

To verify that this is a resync for this instance of the resource manager, your exit program should check that the qualifier passed is the one that is currently in use. If it is not, the exit program should ignore the resync and set a return code of UERFHOLD, to indicate that CICS should keep the disposition of the unit of work.

Parameter 9
Address of a 4-character field containing the next transaction code. If the transaction ended with an EXEC CICS RETURN without specifying the next

transaction code, the addressed field is set to nulls; otherwise, it is set to the value specified by the application. See note 2.

Parameter 10
The address of operation byte 2, which contains the following flags:

UERTONLY
(X'80') Perform a single-phase commit. (No recoverable resources other than those owned by the resource manager being invoked have been updated during the current UOW.)

UERTELUW
(X'40') Perform a single-phase commit. (The resource manager was in read-only mode throughout the current UOW.)

Your exit program should examine the flags set both in this byte and in operation byte 1 (see parameter 1), to determine what action is expected of it.

Note:

1. Parameters 2 through 8 contain values only if the CICS syncpoint manager call is prompted by the issue of an EXEC CICS RESYNC command after a session or system failure, and operation byte 1 contains the bit settings UERTCOMM or UERTBACK. Otherwise, they are set to X'00' (hexadecimal zero). For programming information about the EXEC CICS RESYNC command and about the completion of the syncpointing procedure following a system failure, refer to RESYNC ENTRYNAME in Reference > System programming.

   Note that parameters 2 through 8 describe the environment of the **original** task (not of the task that is currently driving the TRUE).
2. Unless the UERTLAST bit is set in operation byte 1, parameter 9 is a zero address. Although for a call prompted by an EXEC CICS RESYNC call, the UERTLAST bit will be set on, in this case the next transaction code does not apply and so Parameter 9 addresses a field set to nulls.

CICS task manager parameters:

The CICS task manager's parameter list contains one or two entries, depending on the reason for the call to the TRUE: on start-of-task calls, the parameter list contains one entry, while on end-of-task calls, it contains two.

Each entry consists of an address, and the end of the parameter list is indicated by the top bit of the address being set.

The significance of the parameters is as follows:

Parameter 1
The address of a single byte with bit definitions indicating the reason for the call:

UERTSOTR
(X'40') Start of CICS task

UERTEOTR
(X'80') End of CICS task.

Parameter 2
This parameter is passed only on end-of-task calls. It is the address of a 4-character field which contains the next transaction code specified on the EXEC CICS RETURN command. If the transaction ends with an EXEC CICS RETURN without specifying a next transaction, this field is set to nulls.

The schedule flag word should be set during the start-of-task call if you want your task-related user exit program to be invoked unconditionally by the CICS syncpoint manager.

CICS termination manager parameters:

All task-related user exit programs that have been enabled with the SHUTDOWN option of the EXEC CICS ENABLE command, and started, are invoked at CICS termination to allow them to do the clean-up processing that is appropriate to the type of termination. At CICS termination, the address of a one-byte termination code is passed to your exit program.

The code may consist of any of the following bit settings:

UERTCORD
(X'80') CICS orderly shutdown

UERTCIMM
(X'40') CICS immediate shutdown

UERTCABY
(X'20') CICS abend, retry possible, TCBs dispatchable

UERTCABN
(X'10') CICS abend, retry not possible, TCBs dispatchable

UERTOPCA
(X'01') CICS abend, retry not possible, TCBs not dispatchable.

For further information about shutdown TRUEs, see "Coding a program to be invoked at CICS termination" on page 370.

CICS context management parameters:

CICS context management parameters are those parameters that CICS passes when a task-related user exit (TRUE) signals that CICS is to be called for CICS context management.

If the context management bit in the schedule word is set for the current transaction, CICS context management calls the exit program whenever the transaction issues a non-terminal-related **EXEC CICS START** command. The exit program is not called for terminal-related **EXEC CICS START** commands.

When called, the exit program is passed the following parameter list, which is mapped by the DFHUECON DSECT:

UECON_EXEC_PLIST_PTR
The address of a copy of the parameter list passed to the **EXEC CICS START** command. Your task-related user exit program can use this address to access, for example, the data passed on the FROM parameter of the START command, or the name of the transaction to be started. The **EXEC CICS START** parameter list is described by the DFHICUED copy book.

UECON_CORRELATOR_PTR
The address of a 512-byte area in which the exit program can place an ARM workload correlator.

UECON_ICRX_LEN
Reserved for future use.

UECON_ICRX_PTR
Reserved for future use.

The following adapter fields are intended to be used in hierarchical order with the adapter identifier specifying the most general information and adapter data 3 specifying the most specific information. This order provides a uniform manner to identify and isolate tasks.

UECON_ADAPTER_ID_PTR
The address of a 64-character area in which the exit program can pass the data to be placed into the origin data adapter identifier field. Use the same value for all instances of the adapter; for example, the product identifier for the owner of the adapter. If an adapter does not specify an identifier in this area then none of the other adapter data is set.

UECON_ADAPTER_DATA1_PTR
The address of a 64-character area in which the exit program can pass the data to be placed into the origin data adapter data 1 field. This field can be used to identify the server to which the adapter instance (which might be one of many) is connected.

UECON_ADAPTER_DATA2_PTR
The address of a 64-character area in which the exit program can pass the data to be placed into the origin data adapter data 2 field. This field can be used to identify the instance of the adapter task that is starting the task with the START command.

UECON_ADAPTER_DATA3_PTR
The address of a 64-character area in which the exit program can pass the data to be placed into the origin data adapter data 3 field. This field can contain details to identify the reason that the adapter instance started this particular task with the START command.

UECON_FLAGS
The address of a single byte with bit definitions that indicate to the adapter whether any adapter data it sets is placed in the origin data of the task being started.

UECON_ADAPTER_DATA_ON
(X'80') indicates that the adapter is permitted to set origin data adapter data fields.

If the adapter returns values for the adapter data when this flag is not set, the values are ignored. Only the first (origin) adapter for a task, or a set of tasks, is permitted to set the origin data adapter fields.

For remote transactions, the adapter data that applies to the mirror transaction when it is first attached is used for all subsequent START commands that use that mirror, regardless of what is set on their individual START commands.

The DFH$APDT adapter tracking sample TRUE demonstrates how you can use adapter data fields for transaction tracking. For more details, see Adapter tracking sample task-related user exit program (DFH$APDT).

CICS switch application environment parameters:

There are no explicit parameters when CICS calls a task-related user exit for a `switch application environment` call.

Before CICS stops using an open TCB, the TRUE is run on this TCB and the TRUE must release any resources associated with this TCB. The CICS transaction may continue and the TRUE may be called again on a different open TCB, and may then associate its resources with this new open TCB.

For example, a TRUE enabled as CONCURRENCY(REQUIRED) API(CICSAPI) and called from a CICS Java™ application will run on a T8 open TCB. When the Java application completes, if the TRUE has expressed interest in `switch_application_environment` events, the TRUE is called to release its resources from the T8 open TCB. For a subsequent end of task syncpoint, the TRUE will be invoked on an L8 TCB and can associate its resources with the L8 TCB in order to execute its syncpoint processing

CICS EDF build parameters:

On EDF invocations, the address contained in register 1 of the calling program's RSA points to the UEPEDFRM DSECT.

The DSECT contains the following fields:

UEPEDFR1
The address of the application's R1 parameter list.

UEPEDFFI
The input flag byte. When a task-related user exit is invoked by EDF, UEPEDFFI can take one or more of the following bit settings:

UEPEDFRQ
(X'80') “About to Execute” invocation.

UEPEDFRS
(X'40') “Command Execution Complete” invocation.

UEPEDFRA
(X'20') About to display command to EDF.

UEPEDFRC
(X'10') Command has been displayed to EDF.

UEPEDFSC
(X'08') EDF user has changed the screen.

UEPEDFWS
(X'04') EDF user has changed working storage.

UEPEDFNO
(X'01') EDF user has requested NOOP.

UEPEDFFO
The output flag byte. If the task-related user exit requires, it can set the UEPEDFFO flag byte to indicate to EDF what action the task-related user exit wants EDF to take. It can take the following values:

UEPEDFDF
(X'80') Take default CICS action. (EDF screen contains the uninterpreted caller's R1 parameter list.)

UEPEDFND
(X'40') Do not display command to EDF.

UEPEDFRD
(X'20') Redisplay command to EDF.

UEPEDFDL

EDF screen attributes. These are for information only: the task-related user exit program cannot change these fields.

UEPEDFPS (halfword binary)

Page size (number of lines).

UEPEDFLS (halfword binary)

Line size.

UEPEDFMP (halfword binary)

Maximum number of pages.

UEPEDFPA

The address of the EDF display data parameter list, supplied by the task-related user exit. The display data parameter list is composed of alternating pairs of attribute-byte addresses and data-field addresses. Attribute bytes refer to the line of display data pointed to by the data-field addresses. The data field must be the same size as the value specified in UEPEDFLS. The display data is in the format shown in Figure 12.

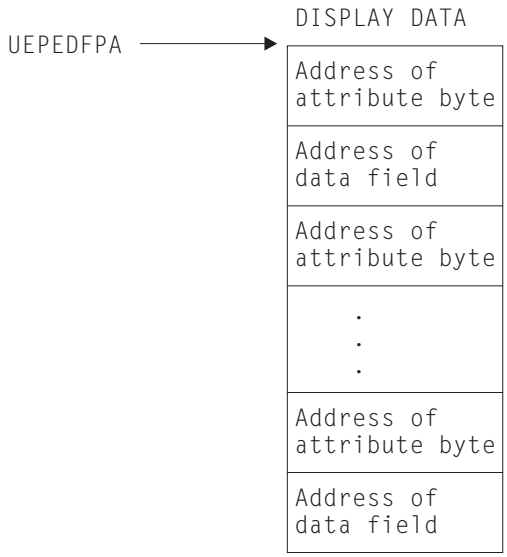


Figure 12. Display data parameter list

Note:

1. CICS provides a list of named standard attribute bytes that you may want to use. These standard attribute bytes are contained within DFHBMSCA, which must be copied into your program. For programming information, including a list of the attribute bytes and their meanings, refer to BMS-related constants, in the *CICS Application Programming Reference* manual.
2. The high-order bit must be set **on** in the last address, to indicate to EDF that this is the last address.

Summary of the task-related user exit parameter lists

Figure 13 on page 358 shows, in diagrammatic form, the relationships between the parameter lists that are discussed in the preceding sections.

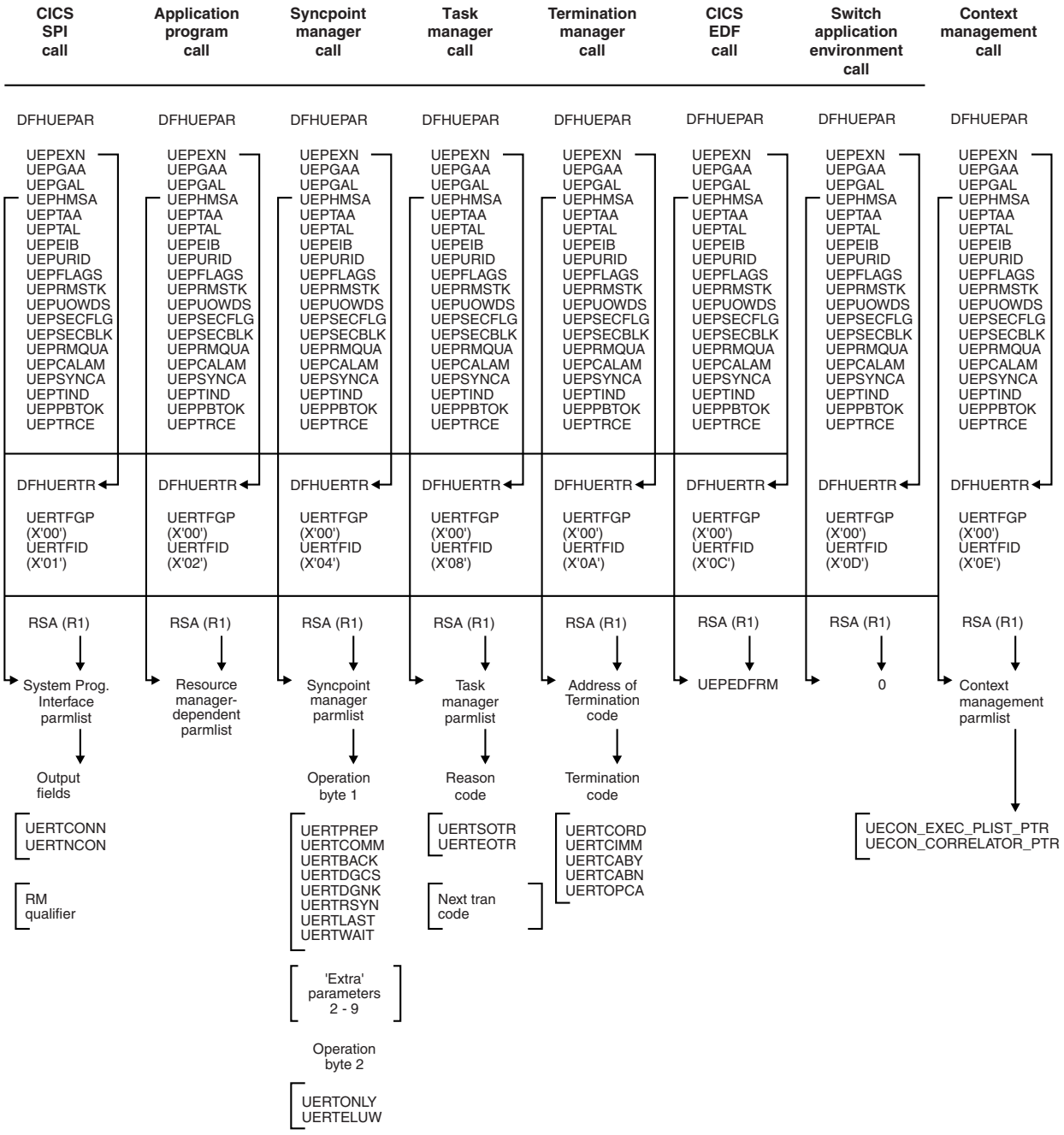


Figure 13. Task-related user exit parameter lists

The schedule flag word

The schedule flag word is a fullword indicator that the task-related user exit program uses to control its own invocation. It is also used by CICS to schedule the first invocation of a task-related user exit program.

The schedule flag word is accessed by the address parameter UEPFLAGS of DFHUEPAR. There is a unique schedule flag word for each association between a CICS task and the ENTRYNAME specified when a task-related user exit program is enabled.

The default setting of the schedule flag word is for application program requests (that is, the last two bytes are set to X'0004').

Table 15. Format of the schedule flag word

Byte	Setting	EXEC CICS ENABLE option	Comments
0	—	—	Reserved
1	—	—	Reserved
2			
UEFDCON	UEFMCON (X'40')	—	Bit mask for context management
UEFDSWAE	UEFMSWAE (X'20')	—	Bit mask for switch application environment
UEFDFEDF	UEFMFEDF (X'10')	FORMATEDF	Bit mask for EDF invocation
UEFDCTER	UEFMCTER (X'04')	SHUTDOWN	Bit mask for termination manager
UEFDTASK	UEFMTASK (X'01')	TASKSTART	Bit mask for task manager
3			
UEFDSYNC	UEFMSYNC (X'10')	—	Bit mask for syncpoint manager
UEFDAPPL	UEFMAPPL (X'04')	—	Bit mask for application program
UEFDSPI	UEFMSPI (X'02')	SPI	Bit mask for SPI

The bit settings of the schedule flag word specify which programs invoke your task-related user exit program. For example, if an exit program is to be invoked by the CICS task manager, the CICS syncpoint manager, **and** an application program, the last two bytes of the schedule flag word should be set to X'0114'. If an exit program is to be called by the CICS task manager and an application program only, the last two bytes of the flag word should be set to X'0104'. Before the exit program is first called by a task, CICS sets the API flag bit on.

The third column of the table shows the option of the EXEC CICS ENABLE command, if any, that can be used to set the bit for each type of invocation. (How to use options of the EXEC CICS ENABLE command to cause a task-related user exit program to be invoked for specific types of call is described in "Enabling for specific invocation-types" on page 376.)

Before returning from any call, the task-related user exit can change the bit settings of the flag word to register its need to be invoked by a different CICS management service, or to register lack of interest in a service by setting the relevant flag bit to zero.

For example, a task-related user exit may be called by an application program that needs to access a non-CICS recoverable resource. When the exit program is first called, the API bit is set on by CICS. If the calling program then issues a request to update a record, the exit program sets the syncpoint manager bit on in the schedule flag word. When the calling application program subsequently issues a syncpoint command, or when end-of-task is reached, the CICS syncpoint manager calls the exit program.

Note: CICS sets the syncpoint manager bit off after every call to the syncpoint manager. This is to avoid the CICS syncpoint manager invoking the task-related user exit program for a unit of recovery during which the exit program did no

recoverable work. The syncpoint manager bit must therefore be set on whenever the exit program performs any recoverable work.

If you set the task manager bit in the schedule flag word on, CICS invokes your task-related exit program at the end of this task. (Note that, if you want your exit program to be called at the **start** as well as at the end of a task, you must specify TASKSTART on the EXEC CICS ENABLE command for the TRUE. This causes the TRUE to be invoked at the start and end of **every** task.)

If the last two bytes of the schedule flag word are set to X'1000', this indicates that the task-related user exit is interested in being invoked by EDF to format requests for display. This schedule flag bit UEFDFEDF is set **on** either by the EXEC CICS ENABLE FORMATEDF command, or by the task-related user exit. Unlike other schedule flag bits, there are restrictions on when the task-related user exit can register a lack of interest in EDF (that is, restrictions on when UEFEDFDF can be set off). Once a task-related user exit has formatted the initial screen for EDF to display on “About to Execute” or “Command Execution Complete”, CICS does not allow it to set the EDF bit UEFDFEDF off until the screen build cycle is complete.

Register handling in the task-related user exit program

When you write your task-related user exit program you must understand the different register types; CICS registers, the calling program's registers and RMI registers. CICS registers are used by the CICS management module that handles task-related user exits. The calling program's registers are registers used by the calling program, and are addressed by parameter UEPHMSA of DFHUEPAR. RMI registers are used by the called resource management interface (RMI).

Saving CICS registers

Start your task-related user exit program by saving the contents of the CICS registers. Register 13 addresses an 18-word area into whose fourth and subsequent words your exit program must store registers 14 through 12. Three of the saved values have significance, as follows:

- The saved contents of register 14 contain the address within CICS to which the task-related user exit program returns control.
- The saved contents of register 15 contain the address at which the task-related user exit program has been entered.
- The saved contents of register 1 address the parameter list (DFHUEPAR) that is provided by CICS for the task-related user exit program.

Note: As a rule, if you fail to understand the origin or the purpose of a call, you must:

1. Restore any registers that you have used to the state they were in on entry to your code
2. Return to the address contained in CICS register 14.

The calling program's registers

The calling program's registers are stored at the address specified by UEPHMSA of DFHUEPAR. Where the calling program is a CICS management program, for example the sync point manager, the only caller registers that have significance are registers 1 and 15. Register 1 addresses the calling program's parameter list. CICS sets the calling program's register 15 to zero before the task-related user exit program is started. The calling program's register 15 can sometimes be used to

pass responses back to the calling program from the task-related user exit program, depending on the identity of the caller. If the calling program is a CICS management program, and the register is still zero on return, CICS assumes that its call was not understood. If the calling program is an application program, the significance of register settings on return are either described in the documentation of your resource manager or defined locally.

The RMI registers

CICS does not support 64 bit addressing for applications, but TRUEs can start RMI programs that use storage at addresses which are only available when running on 64 bit architecture machines. The CICS memory dump formatter shows the contents of the 64 bit General Purpose Registers captured when an abend occurs.

Addressing-mode implications

A task-related user exit (TRUE) program is invoked in the AMODE of the caller, unless you specify the `LINKEDITMODE` option when you enable the exit program.

The `LINKEDITMODE` option enables the task-related user exit program in its link-edit AMODE. Therefore, if the TRUE is link-edited AMODE 31 and is enabled with the LINKEDITMODE option, it can be placed above 16 MB but below 2 GB. For programming information about the LINKEDITMODE option of the EXEC CICS ENABLE command, see ENABLE PROGRAM command in Reference > System programming.

Important: Do not use the LINKEDITMODE option when the TRUE is link-edited AMODE 24. This combination forces the TRUE to always run AMODE 24, which is undesirable for the following reasons:

- An AMODE 24 TRUE cannot be invoked from a transaction that is running with TASKDATALOC(ANY). The result is an AEZB abend.
- Enabling an AMODE 24 TRUE program for task start causes CICS to force all transactions to run with TASKDATALOC(BELOW).
- On a CICS termination call, if CICS detects that the TCA the TRUE is running under is above the 16 MB line, CICS ignores the LINKEDITMODE option and invokes the TRUE in AMODE 31. This is because for some types of termination, such as a cancel, the TCA under which the TRUE will run is not predetermined.

The recommendation for TRUEs is as follows:

- Write the TRUE so that it can always run AMODE 31.
- Link-edit the TRUE AMODE 31.
- Enable the TRUE with the LINKEDITMODE option.

AMODE 64 TRUEs are not supported.

If the task-related user exit program is not enabled with the LINKEDITMODE option of EXEC CICS ENABLE, it is invoked in the AMODE of the caller. For example, for an application request, if the application is AMODE 24 at the time of the DFHRMCAL request, the task-related user exit program is invoked in AMODE 24. For this reason, task-related user exit programs that are enabled without the LINKEDITMODE option must reside below the 16 MB line.

Exit programs and the CICS storage protection facility

When you are running CICS with the storage protection facility, there are two points you must consider for task-related user exits; the execution key in which your task-related user exit programs run and the storage key of data storage obtained for your exit programs.

Execution key for task-related user exit programs

When you are running with storage protection active, CICS always starts task-related user exit programs in CICS key. Even if you specify EXECKEY(USER) on the program resource definition, CICS forces CICS key when it passes control to the TRUE. However, if a task-related user exit program itself passes control to another program (through a link or transfer-control command), the program starts according to the execution key (EXECKEY) defined in its program resource definition.

Important: You must specify EXECKEY(CICS) when defining both task-related user exit programs, and programs to which an exit program passes control.

Data storage key for task-related user exit programs

The storage key of storage used by task-related user exit programs depends on how the storage is obtained:

- Global or local work areas specified when an exit program is enabled, are always in CICS key.
- Any working storage obtained for the exit program is in the key set by the TASKDATAKEY of the transaction under which the exit program is started.
- Task-related user exit programs can use EXEC CICS commands to obtain storage by issuing:
  - Explicit EXEC CICS GETMAIN commands
  - Implicit storage requests as a result of EXEC CICS commands that use the SET option.

  The default storage key for storage obtained by EXEC CICS commands is set by the TASKDATAKEY of the transaction under which the exit program is started.

Recursion within a task-related user exit program

The task-related user exit can invoke itself recursively.

It can do this, for example, by issuing a DFHRMCAL call to its own entry name (as specified on the EXEC CICS ENABLE command). It can also be entered recursively if it performs an EXEC CICS SYNCPOINT when it is interested in SYNCPOINT invocations.

Purging tasks

The operation of task purging when control is within a task-related user exit depends on the PURGEABLE option on the **ENABLE PROGRAM** command.

If the PURGEABLE option is *not* specified:

- Before passing control to a task-related user exit program, CICS inhibits:
  - The ability to purge tasks
  - The monitoring of runaway tasks
- While control is in a task-related user exit program:

– Purge requests are deferred until control is returned from the task-related user exit program.
– Monitoring of runaway tasks is inactive.
– Force purge requests are actioned.

If the PURGEABLE option *is* specified, before passing control to a task-related user exit program CICS inhibits the monitoring of runaway tasks but not the ability to purge tasks. While control is in a task-related user exit program:

- Purge requests are actioned.
- Force purge requests are actioned.
- Monitoring of runaway tasks is inactive.

Wait states in your task-related user exit program

By default, tasks that are active in a task-related user exit and have entered a CICS wait state cannot be purged - only force purge can be used. However, if a task-related user exit is enabled with the PURGEABLE option, a task can be successfully purged from a wait within the task-related user exit.

If this option is to be used, the task-related user exit program must be written to process correctly a purged response from the wait. See ENABLE PROGRAM command in Reference > System programming for more information.

Using CICS services in your task-related user exit program

You can invoke CICS services by issuing CICS API commands in your exit program.

However, you should take note of the following:

- If your program is invoked because of a CICS abend, it must not use any CICS services. See "Coding a program to be invoked at CICS termination" on page 370.
- EXEC CICS commands that cause an XCTL (either directly or implied)—for example, EXEC CICS XCTL or EXEC CICS SHUTDOWN—must never be used.
- DFHEIENT and DFHEIRET must be in your program. *But see the note about not using DFHEIENT in abend invocations, in "Limitations of task-related user exits during CICS shutdown" on page 371.* For further details of the DFHEIENT and DFHEIRET macros, see DFHECALL macro, in the *CICS Application Programming Reference* manual.
- If your exit program entry point is immediately followed by an occurrence of a DFHEIENT macro, inserted either implicitly by CICS or explicitly in the program, then the expansion of the DFHEIENT macro stores incorrect values at DFHEIBP and DFHEICAP. Your code can subsequently correct this by copying UEPEIB into DFHEIBP, reloading the EIB base register (DFHEIBR) from UEPEIB, and setting DFHEICAP to X'80000000'. For example,

```
TESTPROG DFHEIENT CODEREG=2,EIBREG=11,DATAREG=10
         USING DFHUEPAR,1
         MVC   DFHEIBP,UEPEIB             Get correct EIB address
         L     DFHEIBR,UEPEIB             Reload EIB base register
         MVC   DFHEICAP,=X'80000000'
```

  Note that the entry point of a program does not have to be at the start of the program and can be positioned after the DFHEIENT macro.

- The DFHEIENT macro allocates dynamic storage to be mapped by the DFHEISTG DSECT. You must return to CICS by means of the DFHEIRET macro, which frees the dynamic storage.
- Command-level calls use registers 0, 1, 14, and 15.
- Do not issue a syncpoint in start-of-task, end-of-task, or syncpoint invocations.
- On each invocation of a task-related user exit program, a new EXEC environment is created, even when the program is being invoked from the same task. This means that CICS operations, such as browse of a resource definition table, cannot be continued from one invocation of the exit program to the next.

Using channels and containers

Task-related user exit programs cannot access channels and containers created by application programs. They can, however, create their own channels and pass them to programs which they call.

For information about channels and containers, see Enhanced inter-program data transfer: channels as modern-day COMMAREAs, in the *CICS Application Programming Guide*.

Assembler programs and LEASM

Assembler programs translated with the LEASM option cannot be used as task-related user exit programs.

LEASM is used to produce Language Environment conforming main programs in assembler. For information about the LEASM translator option, see LEASM in Translator options in Developing applications.

Work areas

When you use the **EXEC CICS ENABLE PROGRAM** command to identify a task-related user exit program to CICS, you can specify that the program has access to one local work area and one global work area.

The global work area

As with a global user exit program, a task-related user exit program can have its own global work area, or it can share a work area that is owned by another exit program.

When you issue the **ENABLE PROGRAM** command to define the exit, you can specify the GALENGTH and GALOCATION options to make CICS provide a global work area for the exit program. GALENGTH specifies the length of the global work area in bytes, and GALOCATION specifies whether it is in 24-bit storage or 31-bit storage. Alternatively, you can specify the GAENTRYNAME option to name another currently enabled user exit, and your exit program shares the global work area that CICS has already provided for the named exit.

The global work area is associated with the exit program rather than with the exit point. For ease of problem determination, the global work area should be shared only by exit programs that are invoked from the same management module or domain. The address and length of the global work area are addressed by parameters **UEPGAA** and **UEPGAL** of the DFHUEPAR parameter list, which is described in "DFHUEPAR standard parameters" on page 8. If a user exit program does not own a global work area, UEPGAA is set to zero.

Application programs can communicate with user exit programs that use or share the same global work area. The application program uses the **EXEC CICS EXTRACT EXIT** command to obtain the address and length of the global work area.

The local work area

A local work area, also known as a task work area, is a work area with the following characteristics:

- Associated with a single task
- Lasts only for the duration of the task
- Is for the use of a single task-related user exit program

The local work area can be thought of as a logical extension to the transaction work area (TWA, TWACOBA) that is exclusively for the exit program's use.

When you issue the **ENABLE PROGRAM** command to define the exit program, you can specify the TALENGTH option to make CICS provide a local work area for each task that uses the exit. CICS allocates the work area and releases it at task end. If the task-related user exit program was enabled with the LINKEDITMODE option on the **ENABLE PROGRAM** command, and the exit program was link-edited in AMODE(31), the local work area is located in 31-bit storage. If you did not specify the LINKEDITMODE option, or if the task-related user exit program is link-edited in AMODE(24), the local work area is located in 24-bit storage.

The address and length of the global work area are addressed by parameters UEPTAA and UEPTAL of the DFHUEPAR parameter list, which are described in "DFHUEPAR standard parameters" on page 8.

Running an exit program to be started by the CICS SPI

If your task-related exit program has the SPI option of the **EXEC CICS ENABLE PROGRAM** command specified (or your program has the SPI bit-mask in the schedule flag word), your program is started when you use the **EXEC CICS INQUIRE EXITPROGRAM** command to query whether the exit program is connected to its resource manager, and its entryname qualifier.

About this task

For information about the INQUIRE EXITPROGRAM command, see INQUIRE EXITPROGRAM.

Procedure

1. When started for SPI calls, your exit program indicates whether it is connected to its external resource manager, by returning the appropriate value in the first field addressed by the caller parameter list.
   - The following values are returned:

     UERTCONN
     (X'80') The exit is connected to its resource manager.

     UERTNCONN
     (X'40') The exit is not connected to its resource manager.
   - Returns the resource manager qualifier; that is, the entryname qualifier, as returned by the UEPRMQUA parameter of an API call, and used on an **EXEC CICS RESYNC** command, in the second field addressed by the caller parameter list.

Typically, both pieces of information are kept in the global work area of the exit program. The caller parameter list for SPI calls is described in "CICS SPI parameters" on page 350.

2. The RMI SPI call permits a task-related user exit to be called by long-running monitor tasks, even if it has been disabled and reenabled since it was last called by the task. All other types of RMI call fail in this situation. When started for an SPI call, your exit program must not rely on the contents of the task local work area. If the exit has been disabled and reenabled, a new version might have been loaded, which might have a different mapping of the task local work area. The long-running task, however, is running with the original task local work area allocated to it on its first call.

Coding a program to be started by the CICS sync point manager

All task-related user exit programs can be started by the CICS sync point manager.

About this task

An exit program must "schedule" the sync point manager by setting the sync point manager bit-mask in the schedule flag word. The bit-mask must be set after every piece of recoverable work to ensure that the CICS sync point manager calls the exit program during sync point processing. The identification of the current unit of recover, or unit of work, is addressed by the 8-byte field UEPURID. This field is available on all invocations of your exit program in which recoverable actions are possible, for example, application calls and subsequent sync point manager calls.

Increasing efficiency: single-update and read-only protocols

If your resource manager can perform a single-phase commit, you can increase the efficiency of your system through CICS single-update and read-only protocols.

Single-update protocol:

Many CICS transactions use only one external resource manager. In this case, a single-phase commit is in appropriate.

The benefits of a single-phase commit are:

- The resource manager can reduce from two to one the number of log forces required for transactions.
- The number of transaction-related log records written by CICS is reduced.
- A path length reduction is achieved, because the TRUE is invoked only once at the syncpoint, rather than twice.

To take advantage of these benefits, your task-related user exit program must indicate to CICS that the resource manager understands the single-update protocol, and that it (the TRUE) can process a syncpoint call to perform a single-phase commit. It indicates this by setting the UEPSUPDR flag in the field pointed to by UEPSYNCA in the DFHUEPAR parameter list. It must do this every time it sets the syncpoint manager bit in the schedule flag word.

If the exit program has set the UEPSUPDR flag, then, when the syncpoint manager next invokes the TRUE, it informs it whether the resource manager is the only one to have updated resources in the current UOW. It does this by means of the

UERTONLY bit (in operation byte 2 of the syncpoint manager's parameter list); if this is set on, then the resource manager can be asked to perform a single-phase commit.

Read-only protocol:

Similar gains in efficiency can be made if the resource manager is in read-only mode throughout the current unit of work (UOW).

Again, a single-phase commit is appropriate. To benefit, the resource manager must return to the TRUE a flag indicating whether the UOW is read-only or not. The flag may show either the "history" of the UOW so far (for example, so far it is read-only), or whether the current request is read-only. In turn, the TRUE must update the UEPREADO flag in the DFHUEPAR parameter list with the history of the UOW so far. That is, it must set UEPREADO initially, but unset it as soon as the UOW contains updates. (Once UEPREADO has been unset, CICS ignores any subsequent setting of the flag during the current UOW, and treats the UOW as containing updates.)

At the end of the UOW, if the UEPREADO flag is still set, the syncpoint manager invokes the TRUE with instructions to issue a single-phase commit to the resource manager (by setting the UERTELUW bit on).

Return codes

When a task-related user exit program is called by the CICS syncpoint manager, the return codes it is able to set depend on the reason it was called.

Table 16 shows the relationship between the request flags in the syncpoint manager's parameter list and the TRUE return codes. (The CICS syncpoint manager parameters are described in "CICS syncpoint manager parameters" on page 351.)

Table 16. Valid return codes for a TRUE invoked by the CICS syncpoint manager

Request-type	Return codes	Meaning
UERTPREP	UERFPREP	Phase 1 of 2-phase commit successful
UERTPREP	UERFBACK	Phase 1 of 2-phase commit unsuccessful
UERTWAIT	None	Not applicable
UERTCOMM	UERFDONE	Phase 2 of 2-phase commit successful
UERTCOMM	UERFHOLD	Phase 2 of 2-phase commit unsuccessful
UERTBACK	UERFDONE	Backout successful
UERTBACK	UERFHOLD	Backout unsuccessful
UERTONLY	UERFOK	Single-phase commit successful
UERTONLY	UERFBOUT	Single-phase commit failed and backed out
UERTELUW	None	Not applicable

What is expected of your resource manager

If every request from the syncpoint manager prompts a meaningful response from the resource manager, CICS ensures that changes to recoverable resources (such as databases) can be synchronized. That is, either all the changes take effect or all are backed out, even across system failures.

Limitations

Do not update a recoverable CICS resource during a syncpoint call because any changes will not be seen by the CICS syncpoint manager.

Sample code for a TRUE started by the CICS sync point manager

This example pseudocode shows you some of the conditions that a task-related user exit started at a sync point might be required to check.

```
if UERTFID = UERTSYNC then       /* Caller is CICS syncpoint manager */
  select;                        /* Type of syncpoint manager request */
    when (UERTONLY)                          /* ONLY resource manager */
        invoke RM for single-phase commit    /* Single-phase commit */
        if RM single-phase commit succeeded then
          give CICS syncpoint manager 'YES' vote (UERFOK)
        else                           /* Single-phase commit failed */
                                       /* If RM completed backout */
          if RM single-phase commit failed and backed out
            give CICS syncpoint manager 'NO' vote (UERFBOUT)
          else                         /* Don't know what happened */
            put out message and issue transaction abend
          endif
        endif
    when (UERTELUW)                    /* RM read-only for current UOW */
        invoke RM for single-phase commit    /* Single-phase commit */
    when (UERTPREP)       /* Not ONLY resource manager, nor read-only */
        invoke RM for PREPARE  /* Prepare - phase 1 of 2-phase commit */
        select (resource manager vote)
          when (YES)                   /* Phase 1 completed */
            give CICS syncpoint manager 'YES' vote (UERFPREP)
          otherwise
            give CICS syncpoint manager 'NO' vote (UERFBACK)
        endselect
    when (UERTCOMM)             /* Commit - phase 2 of 2-phase commit */
      invoke RM for commit phase 2
      if RM commit succeeded then
        tell CICS sync manager OK (UERFDONE)
      else
        tell CICS sync manager remember could not commit (UERFHOLD)
      endif
    when (UERTBACK)                    /* Backout request */
      invoke RM for backout
      if RM backout succeeded then
        tell CICS sync manager OK (UERFDONE)
      else
        tell CICS sync manager remember could not backout (UERFHOLD)
      endif
    when (UERTWAIT)                    /* CICS indoubt about UOW */
      invoke RM to free thread
         (but maintain locks for UOW and record UOW is indoubt)
  endselect
endif
```

Figure 14. Sample pseudocode for a task-related user exit program to be started by the CICS sync point manager

As described in “Increasing efficiency: single-update and read-only protocols” on page 366, if the UERTONLY bit is set (indicating that the resource manager is the only one to have updated resources) the exit program should cause the resource manager to perform a single-phase commit. If the commit is successful, the exit program should return ‘UERFOK' in register 15; if not, it should return ‘UERFBOUT', meaning that the commit was unsuccessful and the resources were backed out. If the exit program is unsure about the outcome of a single-phase commit, it should abend the task, having saved or displayed any diagnostic information that it considers necessary.

Note that “register 15” in this section refers to the sync point manager's register 15, the fifth word of the area addressed by UEPHMSA.

Similarly, when the UERTELUW bit is set (indicating that the resource manager was in read-only mode throughout this UOW), the exit program should cause the resource manager to perform a single-phase commit. There are no return codes for a UERTELUW call. Because no updates were made, data integrity is not at risk, and therefore no action is taken if the commit fails.

On receiving a request to perform the first phase of a two-phase commit, the resource manager is expected to get into a state where recoverable changes made since the last sync point can be either committed or backed out on demand, even if there is an intervening system failure. For example, buffer contents should be moved to nonvolatile storage. If the resource manager is unable to get into this state, the exit program should return ‘UERFBACK' in register 15, to request backout. Normally, it should return ‘UERFPREP', to indicate a successful phase 1 of a 2-phase commit.

On receiving a request to wait (indicating that CICS is indoubt about the outcome of the UOW), the resource manager should free any task-related resources, such as the thread. However, it should maintain any locks held by the UOW, and record that the UOW is indoubt. See “Enabling for specific invocation-types” on page 376.

On receiving a request to perform the second phase of a two-phase commit, or a request to back out, the resource manager should take the corresponding irreversible step, and have the exit program send the sync point manager a return code: either ‘UERFDONE', meaning that the commit or abend process is complete; or ‘UERFHOLD', meaning that the commit or abend should be resolved later. These return code constants are available to you when you code the macro DFHUEXIT TYPE=RM in your exit program.

If a resource manager cannot understand a call, it should not change the contents of the caller's register 15 before returning to the caller, because it cannot anticipate how the caller interprets the change.

Resynchronization

If a failure occurs between returning from the exit after performing phase 1 of a 2-phase commit and the subsequent phase 2 or back out call, the resource manager must be ready, on restart, to discover the state of the unit of recovery, and to act accordingly.

For programming information about restart resynchronization by using the **EXEC CICS RESYNC** command, see RESYNC ENTRYNAME in Reference > System programming.

If CICS is indoubt about a unit of work, it sends the exit program a request to wait (UERTWAIT). When the status of the indoubt UOW is resolved, CICS initiates a resynchronization task, to inform the exit program of the outcome of the unit of work.

Information about indoubt units of work is retained across both warm and cold starts of CICS. CICS initialization and keypoint management routines recover from the system log all information associating resource managers with outstanding units of recovery, which are resolved automatically when CICS reconnects to the resource managers concerned.

Coding a program to be invoked by the CICS task manager

If your exit program sets the task manager bit in the schedule flag word, it is invoked at end-of-task. If you specify TASKSTART on the EXEC CICS ENABLE command for the TRUE, it is invoked at start-of-task, and (providing it does not unset the task manager bit), at end-of-task too.

About this task

To determine whether a particular invocation is at start- or end-of-task, you can examine the CICS task manager parameters described in "CICS task manager parameters" on page 353. Typically, your program shows interest in task manager events if it needs to save task-related information, such as performance or accounting data, before the task ends.

Exit program limitations

If your task-related user exit program is invoked at end-of-task, you must understand possible limitations on exit program activity at task-detach.

- Do not update any CICS resources at all during a task-detach exit call, because the CICS syncpoint manager is not invoked again for that task. All resources except task-storage have been released by end-of-task.

  Note: Transactions with resource security or command security defined might not run successfully after the terminal has been released. See Resource and command check cross-reference in Securing to determine which resources and commands are subject to security checking. Failure to observe these limitations can result in an ABEND AEY7 - NOTAUTH condition arising.
- It is possible to schedule a new CICS task from your exit program using the **EXEC CICS START** command and to pass data to a new task. However, the **EXEC CICS START** command uses a temporary storage queue to pass data to the new transaction. If this queue is defined as recoverable, it is locked to the detaching task. It is never unlocked, because when the task-detach exit call is made, the resources of the detaching task have already been freed. Use of the PROTECT option causes a different problem: the new task can not be scheduled until the next sync point of the detaching task, but no sync point occurs.
- Do not access remote resources using a task-related user exit program. If you do, you must understand the circumstances in which the function shipping conversation can be terminated.

Coding a program to be invoked at CICS termination

If you specify the SHUTDOWN option when enabling your task-related user exit program, it is invoked at system termination.

About this task

The CICS system termination manager passes the exit program the address of a one-byte code that identifies the type of termination (see "CICS termination manager parameters" on page 354). You can use this invocation of your program to do processing appropriate to the type of termination. For example, at an orderly shutdown you could use it, rather than a PLT program, to shut down the adapter; at a CICS abend you could use it to take special actions, related to the seriousness of the abend.

Limitations of task-related user exits during CICS shutdown

When a task-related user exit (TRUE) is called during shutdown, the capabilities of the exit program are limited.

The nature of CICS abends and operator cancels are such that CICS might not be able to call your exit program at system termination, even if you have specified SHUTDOWN.

The limitations on what your program can do, if called, depend on the type of termination:

Orderly shutdown (UERTCORD)
Your exit program must follow the rules for programs that run during the first quiesce stage of CICS shutdown, when all CICS services are available but programs must not start any new tasks.

Immediate shutdown (UERTCIMM)
Your exit program must do the minimum that is required and return control, so that shutdown can proceed.

CICS abend, retry possible, TCBs dispatchable (UERTCABY)
MVS has flagged the failure as being "eligible for retry". Your exit program must follow the MVS rules for this type of failure, documented in the *z/OS MVS Programming: Authorized Assembler Services Guide*.

Subtasks in the region (that is, task control blocks (TCBs) in addition to the CICS job-step TCB) are still dispatchable, and your exit program can run code under them.

You must not use any CICS services.

CICS abend, retry not possible, TCBs dispatchable (UERTCABN)
MVS has flagged the failure as "not eligible for retry". Your exit program must follow the MVS rules for this type of failure. Your exit program is called from code in the CICS extended subtask abend exit (ESTAE). MVS imposes more restrictions on ESTAE code than on non-ESTAE code.

Subtasks in the region are still dispatchable, and your exit program can run code under them.

You must not use any CICS services.

CICS abend, retry not possible, TCBs not dispatchable (UERTOPCA)
As for UERTCABN, except that subtasks in the region are not dispatchable; your exit program must not try to run code under any TCBs that it might have attached.

Important

In the abend invocations (UERTCABY through UERTOPCA), your exit program must not use any CICS services, including the DFHEIENT call, which performs a CICS GETMAIN. To prevent a DFHEIENT call being issued automatically on each invocation of your program, specify the NOPROLOG translator option; but include in the program source your own DFHEIENT call to be issued on non-abend invocations only. An example of how to code a task-related user exit program to be called at CICS termination is given in Figure 15 on page 373. For further information about coding a DFHEIENT call, see the *CICS Application Programming Reference*.

Sample code for a TRUE invoked at CICS termination

Note that the sample in Figure 15 on page 373 is a multipurpose program—that is, it is coded to be invoked at many task-related user exit invocation points. However, to avoid the need to test for CICS abends in all of your multipurpose TRUEs, it is recommended that you use a separate program for termination invocations.

```
JTRUE1A  CSECT                          TERMINATION TRUE ENTRYPOINT
         STM   14,12,12(13)             Save registers
         USING JTRUE1A,R3
         LR    R3,R15                   Use R3 as base register
         USING DFHUEPAR,R1              Address DFHUEPAR parameter list
         L     R2,UEPEXN
         USING DFHUERTR,R2
         CLI   UERTFID,UERTCTER         CICS Termination call?
         BNE   CONT                     No, so continue
         L     R10,UEPHMSA              Address Host register save area
         USING SA,R10
         L     R5,RSAR1                 Get Caller's R1
         USING DFHCTERM,R5
         L     R5,CTERML                Get termination type
         USING CTERMLIST,R5
         TM    CTERMTYPE,UERTCORD       CICS orderly shutdown?
         BO    CONT                     Yes, so can use CICS services
         TM    CTERMTYPE,UERTCIMM       CICS immediate shutdown?
         BO    CONT                     Yes, so can use CICS services
*        ...
*        ...
*        Insert code here for any processing when CICS is abending
*        (No CICS services should be used)
*        ...
*        ...
         LM    14,12,12(13)             Restore caller's registers
         BSM   0,14                     Return to caller
CONT     DS    0H                       Continue in new CSECT
         LM    14,12,12(13)             Restore callers's registers
         DROP  R3
         USING JTRUE1A,R15              Use R15 as temporary base register
         L     R15,=V(JTRUE1B)          Get address of new CSECT
         BR    R15                      Branch to new CSECT
         DROP  R15
         LTORG
JTRUE1B  CSECT                          POST TEST CSECT
         DFHEIENT
         LR    R4,R1                    Use R4 to address parm list
         USING DFHUEPAR,R4              Address parm list
         L     R5,UEPEXN
         USING DFHUERTR,R5
         MVC   DFHEIBP,UEPEIB
         MVC   DFHEICAP,=X'80000000'
*        .....
*        Insert code here for all types of call other than when CICS
*        is abending
*        (CICS services can be used)
*        .....
EXIT     DS    0H
         DFHEIRET
*
         LTORG
*
DFHCTERM DSECT
CTERML   DS    A
*
CTERMLIST DSECT
CTERMTYPE DS   CL1
*
```

Figure 15. Sample code for a task-related user exit program to be invoked at CICS termination (part 1)

```
DFHEISTG DSECT
*
*        Local working storage for CSECT JTRUE1B
*
RSA      DS    18F                     Register save area
SA       DSECT                         Register save area DSECT
         DS    F
*
RSACB    DS    F       +004
RSACF    DS    F       +008
RSAR14   DS    F       +00C
RSAR15   DS    F       +010
RSAR0    DS    F       +014
RSAR1    DS    F       +018
RSAR2    DS    F
RSAR3    DS    F
RSAR4    DS    F
RSAR5    DS    F
RSAR6    DS    F
RSAR7    DS    F
RSAR8    DS    F
RSAR9    DS    F
RSAR10   DS    F
RSAR11   DS    F
RSAR12   DS    F
         DFHREGS
         DFHUEXIT TYPE=RM
         DFHEISTG
         DFHEIEND
         PRINT NOGEN
         PRINT GEN
         END
```

Figure 16. Sample code for a task-related user exit program to be invoked at CICS termination (part 2)

Using EDF with your task-related user exit program

If your exit program sets the EDF bit in the schedule flag word and EDF is active, the exit program is invoked before and after each API request to format screens for EDF to display.

Communication between the task-related user exit and EDF is controlled by the task-related user exit interface. The command flow between this interface, EDF, and the task-related user exit is summarized in Figure 17 on page 375.

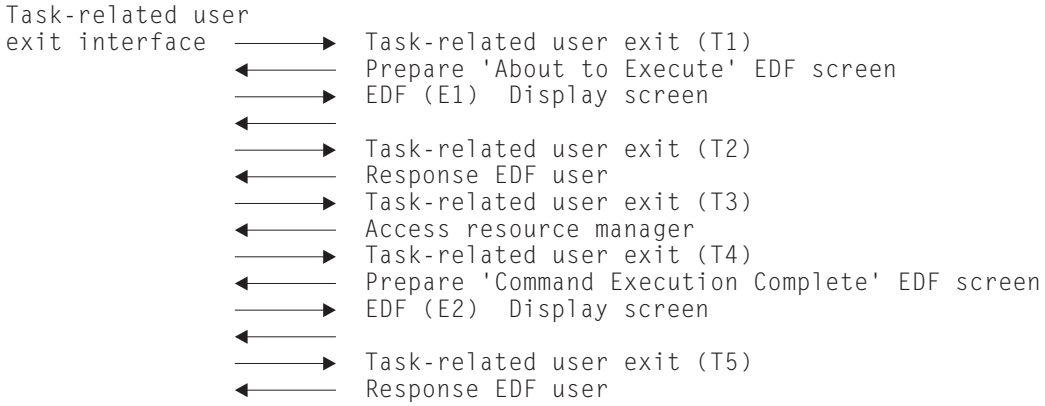


Figure 17. Interface between the task-related user exit and EDF

Table 17 describes each stage of the interface between the task-related user exit and EDF, relating the descriptions to the (T*n*) and (E*n*) expressions in Figure 17.

Table 17. Description of each stage of the task-related user exit/EDF interface

Invocation	Description
(T1)	Task-related user exit invoked to set up its EDF requirements. At this stage the task-related user exit prepares the "About to Execute" screen based on the application request.
(E1)	Using information passed back from the task-related user exit at invocation T1, the task-related user exit interface invokes EDF to display the "About to Execute" screen. EDF sets up the EDF user response, for example, if the user has changed the screen.
(T2)	Task-related user exit is invoked with the EDF user response to the "About to Execute" screen.
(T3)	Task-related user exit invoked to access external resource manager for application request.
(T4)	Task-related user exit invoked to prepare a "Command Execution Complete" screen, based on the result of the application request.
(E2)	Using information passed back from the task-related user exit at invocation T4, the task-related user exit interface invokes EDF to display the "Command Execution Complete" screen. EDF sets up the EDF user response, for example, if the user has changed the screen.
(T5)	Task-related user exit is invoked with the EDF user response to the "Command Execution Complete" screen.

Important

The E1/T2 and E2/T5 cycles can be used repeatedly. This may occur, for example, if the EDF user changes the screen a number of times.

Administering the adapter

Careful use of task-related user exits can allow your application programmers to be unaffected by the invocation of non-CICS resource managers from CICS application programs. Enabling and disabling task-related user exit programs for an installation should be the responsibility of one or more supervisory or master terminal operators.

What you must do before using the adapter

A task-related user exit program must be both enabled and started before it is available for execution.

Procedure

1. Use the **CEDA INSTALL PROGRAM** command to define the task-related user exit program to the system.
2. Use the **EXEC CICS ENABLE PROGRAM** command to enable the task-related user exit program and to define its working storage needs.

Example

```
EXEC CICS ENABLE PROGRAM('EP9')
     TALENGTH(750) GALENGTH(200) SHUTDOWN

EXEC CICS ENABLE PROGRAM('EP9')
     START
```

The first command loads the task-related user exit program, EP9, and causes a global work area of 200 bytes to be obtained and associated with it. To locate the global work area in 31-bit storage, specify the CVDA LOC31 for the GALOCATION option of the command. The first command also schedules the allocation of a local work area of 750 bytes for each task that later starts EP9, and for the invocation of EP9 at CICS termination.

The second command starts the exit program: that is, it makes its entry point capable of being started.

Enabling for specific invocation-types

Use the following options of the **EXEC CICS ENABLE** command to cause your exit program to be started at specific events; INDOUBTWAIT, SHUTDOWN, and SPI.

:

INDOUBTWAIT

Specifies that, at phase 2 sync point time, if CICS is indoubt about the outcome of the UOW, the exit program is to be started with the UERTWAIT verb (wait), instead of a forced definition of UERTCOMM (commit) or UERTBACK (backout). UERTWAIT signifies that CICS does not yet know the outcome of the UOW. In response to a UERTWAIT call, the task-related user exit should start its resource manager to free any task-related resources, such as the thread. However, the resource manager should maintain any locks held by the UOW, and record that the UOW is indoubt.

When CICS receives the outcome of the UOW from its coordinator, a resynchronization task is attached to notify the task-related user exit about the outcome of the UOW.

If CICS is indoubt about the outcome of a UOW for which an external resource manager has requested resynchronization (by using the **EXEC CICS RESYNC** command), CICS waits until the indoubt has been resolved before initiating a resynchronization task.

The effects of **not** enabling a task-related user exit with the INDOUBT keyword are:

- If CICS is indoubt about a UOW, a forced decision is taken and the task-related user exit started with the forced decision.

- If CICS is forced to take a decision because a task-related user exit is not enabled with INDOUBTWAIT, it takes a forced decision for **all** resources updated by the UOW, even if all the other resources are capable of waiting for indoubt resolution. This applies to local resources such as files, and also other RMCs, such as LU6.1, LU6.2, or MRO connections to other systems.
- An inbound RESYNC command from a resource manager that requests resynchronization for a UOW that CICS was indoubt about, results in CICS starting the task-related user exit with a forced decision.

SHUTDOWN
Specifies that the exit program is to be started at CICS shutdown.

SPI Specifies that the exit program is to be started to satisfy EXEC CICS INQUIRE EXITPROGRAM calls that specify the CONNECTST or QUALIFIER options. Use this option to enable user programs to discover whether the exit program is connected to its resource manager, and what its entryname qualifier is.

Note: The exit program can set this option dynamically, by setting the UEFMSPI bit-mask in the schedule flag word.

For programming information about the **EXEC CICS ENABLE PROGRAM** command, refer to ENABLE PROGRAM, in the *CICS System Programming Reference* manual.

The administration routines

As well as enabling task-related user exit programs before they can be used, you should disable them when you have finished using them.

You should prepare procedures (the administration routines) for enabling and disabling your task-related user exit programs, using the EXEC CICS ENABLE and DISABLE commands, and for resynchronizing between sessions or after a system failure. Your enabling routines could be PLT initialization programs or online programs. Your disabling routines could, for example, be started by a TRUE invoked at CICS termination.

The EXTRACT EXIT command obtains the address and the length of a global work area that is owned by, or shared by, a named task-related user exit program.

For programming information about these system commands, and the rules governing them, and also about resynchronization, refer to theIntroduction to System programming commands in Developing system programs manual.

Tracing a task-related user exit program

CICS issues a trace entry just before control is passed to the task-related user exit and just after returning from the exit. You can control these trace entries using the RI option of the CETR trace control transaction or the EXEC CICS SET TRACETYPE command.

Adapter tracking sample task-related user exit program (DFH$APDT)

DFH$APDT is a sample task-related user exit (TRUE) program, which contains adapter data fields that you can use for transaction tracking.

The sample exit program DFH$APDT is supplied in both source and object code. The source is supplied in the hlq.SDFHSAMP sample library, and the executable

form in the hlq.SDFHLOAD load library. You must tailor this sample program before you use it in a production environment.

The DFH$APDT sample TRUE program can be used in one of two ways:

- It can be enabled for TASKSTART, for example by using the command **EXEC CICS ENABLE PROGRAM(DFH$APDT) TASKSTART START** so that it is called at the start and end of every task. The exit sets interest in context management for every task, and is called each time a START command is issued by any subsequent task.
- It can be enabled and started by using the **EXEC CICS ENABLE PROGRAM(DFH$APDT) START** command. The exit can then be started by an application program by using a DFHRMCAL request. The exit sets interest in context management, and is called for each subsequent START command issued by the application program.

When you set interest in context management for a transaction, all subsequent START requests that are issued by the transaction cause the exit to be called and the adapter data fields to be set. In the DFH$APDT sample, these fields are set from constants, but for a real adapter their content is based on context. CICS uses the contents of these fields to populate the adapter data fields in the origin data section of the association data of the task that is being started. This association data is then available for transaction tracking.

Chapter 3. The user exit programming interface (XPI)

The user exit programming interface (XPI) provides global user exit programs with access to CICS services.

Overview of the XPI

The user exit programming interface (XPI) provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs.

The XPI provides opportunities to extend CICS functions beyond the facilities provided in the standard CICS system, but it must be used with care. Any exit programs you write that use this interface must be written by using the following guidance and must be tested carefully to ensure that they cannot cause system errors.

The user exit programs must be in assembler language; the XPI is not provided for other languages. Programs that contain XPI calls must be written to 31-bit standards and must be reentrant.

You must be in primary-space translation mode when you start the XPI. For information about translation modes, see *z/Architecture Principles of Operation*.

- Using the XPI directory functions, you can:
  - Establish a session with an LDAP server. See "The BIND_LDAP call" on page 766.
  - Terminate a session with an LDAP server. See "The UNBIND_LDAP call" on page 775.
  - Remove the contents of all cached search responses for the specified LDAP connection. See "The FLUSH_LDAP_CACHE call" on page 769.
  - Send a search request to a specified LDAP server. See "The SEARCH_LDAP call" on page 773.
  - Browse the results (attributes or entries) returned by the SEARCH_LDAP call. See "The START_BROWSE_RESULTS call" on page 774.
  - Get the next attribute in a series, from an entry returned by the SEARCH_LDAP call. See "The GET_NEXT_ATTRIBUTE call" on page 771.
  - Get the next entry, from a series of entries returned by the SEARCH_LDAP call. See "The GET_NEXT_ENTRY call" on page 772.
  - Retrieve the value associated with an attribute returned by the SEARCH_LDAP call. See "The GET_ATTRIBUTE_VALUE call" on page 770.
  - End the browse session that was started by the START_BROWSE_RESULTS call. See "The END_BROWSE_RESULTS call" on page 768.
  - Release all storage held by the SEARCH_LDAP function. See "The FREE_SEARCH_RESULTS call" on page 769.
- Using the XPI dispatcher functions, you can:
  - Obtain a suspend token for a task. See "The ADD_SUSPEND call" on page 779.
  - Suspend execution of the issuing task. See "The SUSPEND call" on page 782.
  - Resume execution of a suspended task. See "The RESUME call" on page 781.

  - Release a suspend token associated with a task. See “The DELETE_SUSPEND call” on page 781.
  - Request a wait on one or more MVS event control blocks (ECBs). See “The WAIT_MVS call” on page 786.
  - Change the priority of the issuing task. See “The CHANGE_PRIORITY call” on page 780.
- Using the XPI dump control functions, you can:
  - Request a system dump. See “The SYSTEM_DUMP call” on page 790.
  - Request a transaction dump. See “The TRANSACTION_DUMP call” on page 791.
- Using the XPI enqueue domain functions, you can:
  - Enqueue on a named resource. See “The ENQUEUE function” on page 794.
  - Release a resource previously enqueued by an ENQUEUE function call. See “The DEQUEUE function” on page 794.
- Using the XPI kernel domain functions, you can:
  - Inhibit purge for the current task. See “The START_PURGE_PROTECTION function” on page 796.
  - Re-enable purge for the current task. See “The STOP_PURGE_PROTECTION function” on page 796.
- Using the XPI loader functions, you can:
  - Define a new program to the loader domain. See “The DEFINE_PROGRAM call” on page 800.
  - Load a program or, if it is already loaded, obtain its load and entry-point addresses. See “The ACQUIRE_PROGRAM call” on page 798.
  - Locate a program and obtain information about it. See “The IDENTIFY_PROGRAM call” on page 804.
  - Release the storage occupied by a program, or decrement its use count by one. See “The RELEASE_PROGRAM call” on page 805.
  - Delete a program definition from the list of current programs. See The DELETE_PROGRAM call.
- Using the XPI log manager functions, you can:
  - Retrieve information about the activity keypoint frequency of the system. See The INQUIRE_PARAMETERS call.
  - Set the activity keypoint frequency of the system. See The SET_PARAMETERS call.
- Using the XPI monitoring functions, you can:
  - Process a user event-monitoring point. See The MONITOR call.
  - Retrieve the application context data for the issuing task. See “The INQUIRE_APP_CONTEXT call” on page 808.
  - Retrieve the current monitoring data for the issuing task. See The INQUIRE_MONITORING_DATA call.
- Using the XPI object transaction domain functions, you can:
  - Link the current unit of work of a task to an external transaction - see “The IMPORT_TRAN call” on page 813.
  - Perform a sync point on the unit of work of the current task, without referencing an external coordinator - see “The COMMIT_ONE_PHASE call” on page 814.
  - Perform the first phase of the sync point on the CICS unit of work on behalf of an OTS transaction - see The PREPARE call.

  - Perform the second phase of the sync point of an OTS transaction, ensuring that the transaction is committed - see The COMMIT call
  - Roll back an OTS transaction - see The ROLLBACK call.
  - Mark the CICS unit of work so that the global transaction is rolled back at the next sync point - see The SET_ROLLBACK_ONLY call.
  - Associate a link with the current task's unit of work to represent a remote coordinator - see The SET_COORDINATOR call.
- Using the XPI program management functions, you can:
  - Inquire about the attributes of a specified program. See The INQUIRE_PROGRAM call.
  - Inquire about the attributes of the program that is currently running. See The INQUIRE_CURRENT_PROGRAM call.
  - Set selected attributes in the definition of a specified program. See The SET_PROGRAM call.
  - Browse through program definitions, optionally starting at the definition of a specified program. See The START_BROWSE_PROGRAM call, The GET_NEXT_PROGRAM call, and The END_BROWSE_PROGRAM call.
  - Inquire about the settings of the autoinstall function for programs, mapsets, and partitionsets. See The INQUIRE_AUTOINSTALL call.
  - Change the settings of the autoinstall function for programs, mapsets, and partitionsets. See The SET_AUTOINSTALL call.
- Using the XPI state data access functions, you can:
  - Inquire on application system data in the AP domain. See The INQ_APPLICATION_DATA call.
  - Inquire on CICS system data in the AP domain. See The INQUIRE_SYSTEM call.
  - Set CICS system data values in the AP domain. See The SET_SYSTEM call.
- Using the XPI storage control functions, you can:
  - Obtain and initialize storage. See The GETMAIN call.
  - Release storage. See The FREEMAIN call.
  - Inquire about the access-key of an element of storage. See The INQUIRE_ACCESS call.
  - Obtain the start address and length of an element of task-lifetime storage. See The INQUIRE_ELEMENT_LENGTH call.
  - Discover whether CICS is short on storage. See The INQUIRE_SHORT_ON_STORAGE call.
  - Inquire about a tasks task-lifetime storage. See The INQUIRE_TASK_STORAGE call.
  - Cause CICS to switch from a subspace to base space. See The SWITCH_SUBSPACE call.
- Using the XPI trace control function, you can:
  - Write a trace entry to the active trace destinations. See The TRACE_PUT call.
- Using the XPI transaction management functions, you can:
  - Inquire about the environment in which a transaction is running. See The INQUIRE_CONTEXT call.
  - Discover the name of the dynamic transaction routing transaction definition. See The INQUIRE_DTRTRAN call.
  - Discover the current value of the **MXT** system initialization parameter. See The INQUIRE_MXT call.

  - Inquire about a specified transaction class. See The INQUIRE_TCLASS call.
  - Inquire about a specified transaction definition. See The INQUIRE_TRANDEF call.
  - Inquire about an attached transaction. See The INQUIRE_TRANSACTION call.
  - Change the task priority and transaction class of the current task. See The SET_TRANSACTION call.
- Using the XPI user journaling function, you can:
  - Write a record to a CICS journal. See The WRITE_JOURNAL_DATA call.

Important

1. You cannot use all of the XPI calls at every global user exit point. An indication of when these calls cannot be used is in the description of each function call, and in the lists of exit points in Global user exit programs.

   XPI calls are used to start CICS services; using them in the wrong exits causes unpredictable errors in your CICS system.
2. There is a restriction on using the XPI early during initialization. Do not start exit programs that use the XPI functions INQUIRE_MONITOR_DATA, MONITOR, TRANSACTION_DUMP, and WRITE_JOURNAL_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Writing initialization and shutdown programs.
3. These XPI functions are likely to cause the task executing the user exit program to lose control to another task while the XPI function is being run. Therefore the use of XPI functions must be carefully considered, as interrupting the flow of CICS functions might cause problems, such as lockouts, to occur.
4. A global user exit or task-related user exit might be assembled using CICS libraries from one CICS release and make an XPI call on a system that runs a different CICS release. In this situation, whether or not control is successfully transferred from the exit to the correct CICS module to handle that XPI call depends on the combination of CICS releases, and whether the XPI call is a release-sensitive call. For details, see Upgrading.

Making an XPI call

An XPI call has two sets of parameters: input parameters, including the XPI function call and the parameters passed to the call, and output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

To use an XPI macro call, you must include a copy book that defines the input and output parameters. The name of the macro is always of the form DFH*xxyy*X, and the associated copy book has the name DFH*xxyy*Y. For example, the GETMAIN call is part of the storage control XPI. The macro you would use is DFHSMMCX and the associated copy book is DFHSMMCY.

The general format (omitting the assembler-language continuation character) of all XPI calls is:

```
macro-name [CALL],
           [CLEAR],
           [IN,
           FUNCTION(call_name),
           mandin1(value),
           mandin2(value),
           ...
```

```
[optin1(value),]
[optin2(value),]
...]
[OUT,
mandout1(value),
mandout2(value),
...
[optout1(value),]
[optout2(value),]
...
RESPONSE,
REASON]
```

XPI calls follow assembler-language coding conventions:

- The "macro-name" must begin before column 16.
- The continuation lines must begin in column 16.
- There must be no embedded blanks apart from the blank between the macro-name and the first keyword (usually CALL).
- Entries on lines other than the final line must end with a comma.
- Lines other than the final line must have a continuation character in column 72.
- Parentheses around the input and output values are required—and if you use a register reference as an input or output value, it must be enclosed in an inner pair of parentheses, thus: ((R6)).
- For details about how to set the values of the XPI options, refer to "XPI syntax" on page 393.

There are three uses of these XPI functions. You can:

- Clear the parameter list used by the XPI call
- Set up the input parameters
- Make the call to the CICS function.

You can code all of these individually (see "An example showing how to build a parameter list incrementally" on page 392), or include them in a single statement.

Some options are common to all uses of the XPI. They are included in all of the syntax descriptions, but their explanation is given here. The options are CALL, CLEAR, IN, FUNCTION, OUT, RESPONSE, and REASON.

CALL causes code generation that issues a call to the XPI function. If you specify CALL, IN, FUNCTION, and OUT, then code is generated to perform the whole operation of building the parameter list, invoking the function, and receiving the result. You can omit the CALL, but specify IN to build your parameter list incrementally; later you can use CALL with that list, coding CALL, IN, FUNCTION, OUT, and all required options. You can then represent the values of the preset options by an asterisk (*) to show that the value is already present in the list.

Note: If you build your parameter list incrementally, do not specify CLEAR when you finally issue the call, because the CLEAR option sets the parameter list to zeros, which will cause you to lose the preset values.

CLEAR
sets the existence bits in the parameter list (both mandatory and optional parameters) to binary zeros. Each macro has a COPY code, which defines the parameter list by a DSECT consisting of a header section, followed by a set of existence bits, and the parameters themselves. For performance

reasons, the header section and the existence bits only are cleared. The rest of the parameter list remains unchanged.

Note: Failure to clear the parameter list can cause unpredictable results, such as program checks or storage violations. If you are building the parameter list incrementally, specify CLEAR before specifying any parameters. If you are not building the parameter incrementally, specify CLEAR when the CALL is issued.

IN tells CICS that any parameter following the IN option and preceding the OUT option is an input value. It **must** be specified when CALL is specified. If you use the function without CALL to build a parameter list, you can specify IN and some parameter values to store values into your list.

FUNCTION
specifies which function of the macro you require; for instance, GETMAIN or FREEMAIN. It **must** be specified when CALL is specified, and unlike other options, it must always be explicit—you **cannot** code "FUNCTION(*)".

mandin(value)
"mandin" represents an option that becomes mandatory if CALL is specified. "value" may be represented by an asterisk (*) to show that a previous use of the macro has already set the value in the parameter list (see "CALL"). For further details about how to complete "value", refer to the specific function calls in "XPI syntax" on page 393.

OUT tells CICS that any parameter following the OUT option is a receiver field. It **must** be specified when CALL is specified.

Note: The use of the following output parameters with values other than an asterisk (*) is invalid if CALL is not specified.

mandout(value)
"mandout" represents an option that becomes mandatory if CALL is specified. The output is placed in the parameter list if an asterisk (*) is coded, or in the location that you have specified in "value". RESPONSE is a special case of a mandout option (see RESPONSE). For further details about how to complete "value", refer to the specific function calls (see "XPI syntax" on page 393).

optin1,2...; optout1,2....
represent items that are completely optional for **all** forms of the macro; in particular, they do not have to be specified when CALL is specified.

RESPONSE
is a mandatory data area that you define to receive the response from your XPI call. You can use an asterisk (*) to indicate to CICS that the RESPONSE value is to be placed in the parameter list, or you can specify the name of a field in which you want the RESPONSE value to be placed. You need not code the RESPONSE option if you are using the macro without CALL to build a parameter list.

The response from any XPI call is always one of 'OK', 'EXCEPTION', 'DISASTER', 'INVALID', 'KERNERROR', and 'PURGED'. There are standardized names (EQU symbols) for the response code values provided by CICS:

```
xxyy_OK, xxyy_EXCEPTION, xxyy_DISASTER, xxyy_INVALID,
xxyy_KERNERROR, and xxyy_PURGED,
```

where “xxyy” is a prefix derived from the four letters of the relevant macro-name following the string ‘DFH'. Thus for DFHSMMCX the prefix is SMMC; for DFHLDLDX the prefix is LDLD. Equate values with these names are generated when you include the copy book DFHxxyyY for the macro DFHxxyyX. You cannot assume that the arithmetic values of corresponding RESPONSE codes are the same for all macro calls. The meanings of the RESPONSE codes are as follows:

OK The XPI request was completed successfully.

EXCEPTION
The function was not completed successfully for a reason which could be expected to happen, and which may be coded for by a program (for example, TRANSACTION_DUMP, EXCEPTION = SUPPRESSED_BY_DUMPTABLE). Any REASON value may provide more information.

DISASTER
The request has failed completely. You cannot recover from this failure within the user exit program. When this failure occurs, CICS takes a system dump, issues an error message, and sets a ‘DISASTER' response. On receiving this, your user exit program should exit without attempting any further processing. The REASON value for this response, shown only in the trace, may provide more information. There is no REASON value returned to the calling program.

INVALID
You have omitted a mandatory value, or you have supplied an invalid value for an option. You cannot recover from this failure within the user exit program. When this failure occurs, CICS takes a system dump, issues an error message, and sets an ‘INVALID' response. On receiving this response, your user exit program should return to the caller without attempting any further processing. The REASON value for this response, shown only in the trace, may provide more information. This may help you to correct any error in your exit program. There is no REASON value returned to the calling program.

KERNERROR
The kernel has detected an error with the CICS function you are trying to invoke. Either the function you have requested is unavailable or not valid, or there is an error within CICS.

PURGED
The task has been purged, or an interval specified on your XPI call has expired. Examine the REASON code.

Note that if an XPI call other than DFHDSSRX SUSPEND or WAIT_MVS gets this RESPONSE, your exit program should set the return code to ‘UERCPURG' and return to the caller.

If a DFHDSSRX SUSPEND or WAIT_MVS call specifies an INTERVAL and gets this RESPONSE with a REASON of ‘TIMED_OUT', it indicates that the INTERVAL you specified has passed. It is up to you to decide what you do next.

If a DFHDSSRX SUSPEND or WAIT_MVS call specifies an INTERVAL and gets this RESPONSE with a REASON of ‘TASK_CANCELLED', this indicates that the INTERVAL you

specified has not passed but that the task has been purged by an operator or an application. In this case, you must set a return code of 'UERCPURG' and return.

If a DFHDSSRX SUSPEND or WAIT_MVS call does **not** specify an INTERVAL, and gets this RESPONSE with a REASON of 'TASK_CANCELLED' or 'TIMED_OUT', it indicates that the task has been purged by an operator or an application, or by the deadlock time-out facility. In this case, you must set a return code of 'UERCPURG' and return.

You **must not** return the response code 'UERCPURG' to CICS for any other reason. If you attempt to do so, your program will have unpredictable results.

REASON
This is a mandatory data area that you define in order to receive more information about the RESPONSE value. You can use (*) to indicate to CICS that the REASON value is to be placed in the parameter list. On most XPI calls, standardized reason names (EQU symbols) are provided only for RESPONSE values of 'EXCEPTION' and 'PURGED'. The REASON values that accompany responses vary from one XPI function to another, so details are provided with the descriptions of the XPI calls.

REASON is not applicable where RESPONSE was 'OK'. In these circumstances, you should not test the REASON field.

Note: For examples of how to initialize the parameter list, set up parameters, make the call, and receive output parameters, refer to "Global user exit XPI examples, showing the use of storage" on page 388. That section includes both a complete example, and also an example in which each step is executed separately.

Setting up the XPI environment

The exit programming interface (XPI) does not require the usual CICS transaction environment, but you must set up a special exit programming environment before you can use any XPI calls.

If you are going to use any of the XPI functions in an exit program, you must include the following macro in your program, before you issue any XPI calls:

```
DFHUEXIT TYPE=XPIENV
```

The expansion of this macro provides the DSECTs that are used in all XPI calls. It also provides a list of register equates (R0 EQU 0, R1 EQU 1, and so on), that you can use in your exit program. The other fields generated by the macro are used by CICS to perform the XPI call processing. You must not use any of these fields: if you do so, your user exit program will have unpredictable results.

The user exit program must be in 31-bit addressing mode.

XPI register usage

Before you can issue an XPI call from a global user exit program, you must move the contents of the parameter UEPSTACK (the kernel stack entry) of DFHUEPAR to the exit program's register 13.

The XPI function expansion uses registers 0, 1, 14, and 15, so the exit program must save and restore them if necessary around an XPI call.

For an example of how to use EXEC CICS commands and XPI calls in the same exit program, see Appendix G, "The example program for the XTSEREQ global user exit, DFH$XTSE," on page 917.

The XPI copy books

For each XPI function, a copy book provides the DSECTs associated with that function. These DSECTs allow you to map the parameters and the response and reason codes of an XPI call.

You must include in your exit program a COPY statement for each XPI function that you are going to use. The copy book name is the same as the macro name, except that the final letter "X" becomes a letter "Y".

For example, to include the copy book for the XPI function DFHSMMCX, you must include the statement:

```
COPY DFHSMMCY
```

Trace entries for your XPI calls show these parameter lists if you have tracing on for the function you are using.

Reentrancy considerations resulting from XPI calls

During an XPI call, CICS might give control to another task while processing the XPI call. This second task could call the same exit program and make the same XPI call, possibly with different parameter values. In this situation, you must ensure that lockout situations do not occur.

While processing an XPI call, CICS might encounter another user exit point that uses the same user exit program. Therefore, the XPI parameter lists must be built in storage associated with a single invocation of the exit program.

If your exit program is a global user exit, CICS provides it with 1024 bytes of LIFO storage, which is exclusive to a single invocation of your exit program. Your exit program can access this storage using parameter UEPXSTOR of the DFHUEPAR parameter list. Use this storage to base the DSECT provided by the DFHxxyyY copy book when building the XPI parameter list. In this way, the parameters are not corrupted if the exit program is reentered.

Parameter lists for the XPI services provided here do not exceed 256 bytes. The remaining 768 bytes of the UEPXSTOR storage can be used by your exit program for its own purpose. It is expected that the 768 bytes of spare storage will, in most cases, avoid the need for your exit programs to obtain more storage. If you do need to obtain more than the extra 768 bytes provided, obtain it by either a DFHSMMCX FUNCTION (GETMAIN) macro, or an MVS GETMAIN request.

Information to be kept across invocations of an exit program can be stored in the global work area that you can define for an exit program (or group of exit programs). The 1024 bytes of LIFO storage cannot be used for this purpose because it is dynamic.

Release-sensitive XPI call

You can use a release-sensitive XPI call so that the XPI call can execute successfully on all currently supported CICS releases. To do this, replace the **CALL** XPI parameter with the **RELSENSCALL** XPI parameter, and assemble the program. You can use the release-sensitive XPI call alternative with all XPI commands.

The **RELSENSCALL** parameter ensures only that the call to the CICS XPI module that supports that function is successful. You must check that the parameters specified on the XPI call are valid for all the CICS releases on which that call is made. Apart from using a different call parameter, all other XPI syntax rules apply. For more details on the syntax rules, see "XPI syntax" on page 393.

The following example is an XPI GETMAIN call that uses the **RELSENSCALL** parameter:

```
DFHSMMCX RELSENSCALL,                                  -
      CLEAR,                                             -
      IN,                                                -
      FUNCTION(GETMAIN),                                 -
      GET_LENGTH((r6)),                                  -
      SUSPEND(YES),                                      -
      STORAGE_CLASS(USER),                               -
      OUT,                                               -
      ADDRESS((r5)),                                     -
      RESPONSE(*),                                       -
      REASON(*)
```

The following example is the same XPI GETMAIN call that uses the **CALL** parameter:

```
DFHSMMCX CALL,                                         -
      CLEAR,                                             -
      IN,                                                -
      FUNCTION(GETMAIN),                                 -
      GET_LENGTH((r6)),                                  -
      SUSPEND(YES),                                      -
      STORAGE_CLASS(USER),                               -
      OUT,                                               -
      ADDRESS((r5)),                                     -
      RESPONSE(*),                                       -
      REASON(*)
```

For further details about the effect of different CICS releases and release-sensitive calls on user exits, see Changes to global user exits, task-related user exits, and the exit programming interface.

Global user exit XPI examples, showing the use of storage

The following example illustrates the use of the XPI and storage in a global user exit program. It is not a complete program, but merely an example of entry and exit code for any global user exit program, and the use of the XPI function.

The options of the DFHSMMCX macro used in the example are described in "Storage control XPI functions" on page 844.

The example uses the technique of obtaining some storage for this invocation of the program using the XPI GETMAIN call, and then saving the address of this storage in the first 4 bytes of the LIFO storage addressed by UEPXSTOR. In this example, the initialization of the parameter list (using the CLEAR option), the building of the parameter list, and the GETMAIN call occur in a single macro. For details of how to build the parameter list incrementally, and how to separate the CLEAR and the GETMAIN call, refer to "An example showing how to build a parameter list incrementally" on page 392.

```
TITLE 'GUEXPI - GLOBAL USER EXIT PROGRAM WITH XPI'
*************************************************************************
*  The first three instructions set up the global user exit             *
*  environment, identify the user exit point, prepare for the use of    *
*  the exit programming interface, and copy in the definitions that     *
*  are to be used by the XPI function.                                  *
*************************************************************************
*
         DFHUEXIT TYPE=EP,ID=XFCREQ     PROVIDE DFHUEPAR PARAMETER
*                                       LIST FOR XFCREQ IN THE FILE
*                                       CONTROL PROGRAM AND LIST
*                                       OF EXITID EQUATES
*
         DFHUEXIT TYPE=XPIENV           SET UP ENVIRONMENT FOR
*                                       EXIT PROGRAMMING INTERFACE --
*                                       MUST BE ISSUED BEFORE ANY
*                                       XPI MACROS ARE ISSUED
*
         COPY  DFHSMMCY                 DEFINE PARAMETER LIST FOR
*                                       USE BY DFHSMMCX MACRO
*
*************************************************************************
*  The following DSECT maps a storage area you can use to make the      *
*  exit program reentrant by storing the address of the storage you     *
*  acquire in the first four bytes of the 260-byte area provided by     *
*  the user exit handler (DFHUEH) and addressed by UEPXSTOR.            *
*************************************************************************
*
TRANSTOR DSECT                          DSECT FOR STORAGE OBTAINED BY
*                                       GETMAIN
.
.
.
storage declarations
.
.
.
*
*************************************************************************
*  The next seven instructions form the normal start of a global user   *
*  exit program, setting the program addressing mode to 31-bit, saving  *
*  the calling program's registers, establishing base addressing, and   *
*  establishing the addressing of the user exit parameter list.         *
*************************************************************************
*
GXPI     CSECT
GXPI     AMODE 31                       SET TO 31-BIT ADDRESSING
*
         SAVE (14,12)                   SAVE CALLING PROGRAM'S REGISTERS
*
         LR    R11,R15                  SET UP USER EXIT PROGRAM'S
         USING GXPI,R11                 BASE REGISTER
*
         LR    R2,R1                    SET UP ADDRESSING FOR USER
         USING DFHUEPAR,R2              EXIT PARAMETER LIST -- USE
*                                       REGISTER 2 AS XPI CALLS USE
*                                       REGISTER 1
*
```

Figure 18. Global user exit program with XPI (part 1)

```
*************************************************************************
*  Before issuing an XPI function call, set up addressing to XPI        *
*  parameter list.                                                      *
*************************************************************************
*
         L     R5,UEPXSTOR              SET UP ADDRESSING FOR XPI
*                                       PARAMETER LIST
*
         USING DFHSMMC_ARG,R5           MAP PARAMETER LIST
*
*************************************************************************
*  Before issuing an XPI function call, you must ensure that register   *
*  13 addresses the kernel stack.                                       *
*************************************************************************
*
         L     R13,UEPSTACK             ADDRESS KERNEL STACK
*
*************************************************************************
*  Issue the DFHSMMCX macro call, specifying:                           *
*                                                                       *
*   CALL --      the macro is to be called immediately                  *
*                                                                       *
*   CLEAR --     initialize the parameter list before inserting values. *
*                                                                       *
*   IN --        input values follow.                                   *
*                                                                       *
*                FUNCTION(GETMAIN) -- acquire storage                   *
*                GET_LENGTH(120) -- 120 bytes of it                     *
*                SUSPEND(NO) -- don't suspend if storage not available  *
*                INITIAL_IMAGE(X'00')  -- clear acquired storage        *
*                                         to hex zero throughout.       *
*                STORAGE_CLASS(USER)   -- class of storage to be        *
*                                         acquired is user storage      *
*                                         above the 16MB line.          *
*                                                                       *
*   OUT --       output values follow                                   *
*                                                                       *
*                ADDRESS((R6)) -- put address of acquired storage in    *
*                                 register 6.                           *
*                RESPONSE(*) -- put response at SMMC_RESPONSE in        *
*                               macro parameter list.                   *
*                REASON(*) -- put reason at SMMC_REASON in macro        *
*                             parameter list.                           *
*                                                                       *
*************************************************************************
*
         DFHSMMCX CALL,                                                *
               CLEAR,                                                  *
               IN,                                                     *
               FUNCTION(GETMAIN),                                      *
               GET_LENGTH(120),                                        *
               SUSPEND(NO),                                            *
               INITIAL_IMAGE(X'00'),                                   *
               STORAGE_CLASS(USER),                                    *
               OUT,                                                    *
               ADDRESS((R6)),                                          *
               RESPONSE(*),                                            *
               REASON(*)
*                                                                       *
```

Figure 19. Global user exit program with XPI (part 2)

```
*********************************************************************
* Test SMMC_RESPONSE -- if OK, then branch round error handling.    *
*********************************************************************
*                                                                   *
         CLI   SMMC_RESPONSE,SMMC_OK    CHECK RESPONSE AND...
         BE    STOK                     ...IF OK, BYPASS ERROR ROUTINES
*                                                                   *
         .
         .
         .
         error-handling routines
         .
         .
         .
*********************************************************************
*  The next section maps TRANSTOR on the acquired storage.          *
*********************************************************************
STOK     DS    0H
         USING TRANSTOR,R6              MAP ACQUIRED STORAGE
         ST    R6,0(R5)                 SAVE STORAGE ADDRESS IN FIRST
*                                       4 BYTES OF STORAGE ADDRESSED
*                                       BY UEPXSTOR
*
         LA    R5,4(R5)                 ADDRESS 4-BYTE OFFSET
         DROP  R5                       REUSE REGISTER 5 TO BASE ALL
         USING DFHxxyy_ARG,R5           FOLLOWING XPI PARAMETER LISTS
*                                       AT 4-BYTE OFFSET INTO STORAGE
*                                       ADDRESSED BY UEPXSTOR
.
.
.
rest of user exit program
.
.
.
*
*********************************************************************
*  When the rest of the exit program is completed, free the storage
*  and return.
*********************************************************************
*
         DROP  R5                       REUSE REGISTER 5 TO MAP DFHSMMC
         USING DFHSMMC_ARG,R5           XPI PARAMETER LIST
*
         L     R13,UEPSTACK             ADDRESS KERNEL STACK
*
*********************************************************************
*  Issue the DFHSMMCX macro call, specifying:                       *
*                                                                   *
*   CALL --      the macro is to be called immediately.             *
*                                                                   *
*   CLEAR --     initialize the parameter list before inserting values. *
*                                                                   *
*   IN --        input values follow.                               *
*                                                                   *
*                FUNCTION(FREEMAIN) -- release storage              *
*                ADDRESS((R6)) -- address of storage is in register 6.  *
*                STORAGE_CLASS(USER)   -- class of acquired storage was *
*                                         31-bit user storage.      *
*                                                                   *
```

Figure 20. Global user exit program with XPI (part 3)

```
*   OUT --       output values follow                                *
*                                                                    *
*                RESPONSE(*) -- put response at SMMC_RESPONSE in     *
*                               macro parameter list.                *
*                REASON(*) -- put reason at SMMC_REASON in macro     *
*                             parameter list.                        *
*                                                                    *
**********************************************************************
*
         DFHSMMCX CALL,                                              +
               CLEAR,                                                +
               IN,                                                   +
               FUNCTION(FREEMAIN),                                   +
               ADDRESS((R6)),                                        +
               STORAGE_CLASS(USER),                                  +
               OUT,                                                  +
               RESPONSE(*),                                          +
               REASON(*)
*                                                                    *
**********************************************************************
* Test SMMC_RESPONSE -- if OK, then branch round error handling.     *
**********************************************************************
*                                                                    *
         CLI   SMMC_RESPONSE,SMMC_OK    CHECK RESPONSE AND...
         BE    STEND               ...IF OK, BYPASS ERROR ROUTINES
*                                                                    *
         .
         .
         .
         error-handling routines
         .
         .
         .
*
**********************************************************************
*  Restore registers, set return code, and return to user exit handler  *
**********************************************************************
*                                                                    *
STEND    DS    0H
         L     R13,UEPEPSA
         RETURN (14,12),RC=UERCNORM
         LTORG
         END   GXPI
```

Figure 21. Global user exit program with XPI (part 4)

An example showing how to build a parameter list incrementally

This example illustrates a parameter list that is built incrementally. The initialization of the parameter list, using the CLEAR option, the building of the parameter list, and the GETMAIN call are separated into discrete steps.

```
         DFHSMMCX CLEAR
:
         DFHSMMCX GET_LENGTH(100)
:
         DFHSMMCX CALL,                                              *
               IN,                                                   *
               FUNCTION(GETMAIN),                                    *
               GET_LENGTH(*),                                        *
               SUSPEND(NO),                                          *
               INITIAL_IMAGE(X'00'),                                 *
               STORAGE_CLASS(USER),                                  *
```

```
OUT,                                                    *
ADDRESS((R6)),                                          *
RESPONSE(*),                                            *
REASON(*)
```

Important

You must set your parameters using **only** the XPI functions.

XPI syntax

The XPI functions use special syntax. The description of each function defines only the options that are specific to that call.

Options that are applicable to all function calls are described in "Making an XPI call" on page 382. The following argument types are used:

name1, name2,....
Each of these refers to the name of a field of the given size in bytes. "name1" means that the name you specify should be that of a 1-byte field.

literalconst
A number in the form of a literal, for example B'00000000', X'FF', X'FCF4', "0", or an equate symbol with a similar value.

expression
A valid assembler-language expression: a decimal integer, or any arithmetic expression (including symbolic values) valid in assembler language; for example:

```
20; L'AREA; L'AREA+10; L'AREA+X'22'; SYMB/3+20 .
```

(Rn) A register reference. The parentheses shown here are required in addition to those that surround the argument. For example: OPTION((R5)).

block-descriptor
Represents a source of both the data address and the data length fields. A block-descriptor can be either a single value or a double value. The following is the single-value form:

```
OPTION(blkdname)
```

blkdname
The name of a block-descriptor. A pair of contiguous fullwords, in which the first word contains the address of the data, and the second word contains the length in bytes of the data, as a fullword binary value. Register notation is not accepted for this single-value form.

The following is the double-value form:

```
OPTION(addr,len)
```

addr The data address as {namea | (Ra) | aliteral}:

namea The name of a location containing the data address

(Ra) A register containing the data address

aliteral
An address constant literal; for example: A(data).

len The data length as {namel | (Rn) | expression}:

namel The name of a location containing a binary fullword giving the data length in bytes

(Rn) A register, the contents of which specify in fullword binary the number of bytes of data

expression
A decimal integer, or any arithmetic expression, including symbolic values, valid in assembler language; for example:

```
L'AREA ; L'AREA+10 ; L'AREA+X'22' ; SYMB/3+20 .
```

buffer-descriptor
Represents a source of both the data address and the maximum data length fields. Parts of the buffer-descriptor are also reserved to act as receiving fields for output information. A buffer-descriptor can be either a single value or a multiple value. The following is the single-value form:

```
OPTION(bufdname)
```

bufdname
The name of a buffer-descriptor. A group of up to four contiguous fullwords, that represent multiple components of the buffer-descriptor. The fields are interpreted as follows:

- The first word contains the address of the data (input).
- The second word is reserved to receive the current length in bytes of the data, as a fullword binary value (output). If three components are specified, the third component maps to this word. If only two components are specified in the buffer-descriptor, the second component maps to this word.
- The third word contains the maximum length in bytes of the data, as a fullword binary value (input). If three components are specified, the second component maps to this word. If only two components are specified in the buffer-descriptor, this field is not used.
- The fourth word is reserved for use by the XPI.

Register notation is not accepted for this single-value form.

The following is the multiple-value form:

```
OPTION(addr,maxlen,*)
```

addr The data address as {namea | (Ra) | aliteral}:

namea The name of a location containing the data address

(Ra) A register containing the data address

aliteral
An address constant literal, for example, A(data).

maxlen
The maximum data length as {namel | (Rn) | expression}:

namel The name of a location containing a binary fullword giving the maximum data length in bytes

(Rn) A register, the contents of which specify in fullword binary the maximum number of bytes of data

expression
A decimal integer, or any arithmetic expression, including symbolic values, valid in assembler language; for example:

```
L'AREA ; L'AREA+10 ; L'AREA+X'22' ; SYMB/3+20 .
```

* A required parameter to indicate that the parameter list is to be used for the reserved fields. If this parameter is coded, then the required value must be taken from the _N component returned in the buffer-descriptor.

Chapter 4. Customizing data tables using user exits

Three global user exit points are included in data table services. You can supply one or more assembler language programs to be executed at each of these points in order to extend or modify the function provided by CICS.

This section contains Product-sensitive Programming Interface and Associated Guidance Information.

CICS supplies sample user exit programs in the SDFHSAMP library for the XDTRD, XDTAD, and XDTLC global user exits. These programs, which are reproduced in this documentation, describe with samples of coding and data definition sequences the conventions used in user exit programs that are used with shared data tables. These samples are intended only as general guidance and do not define a programming interface.

Programming information about global user exits and how to use them is given in Global user exit programs in Developing system programs.

Note: The EXEC interface user exits XEIIN and XEIOUT, and the file control user exits XFCREQ and XFCREQC, are not started in the file-owning region if a request to access a data table is satisfied by cross-memory services.

Communicating between CICS and shared data table exit programs

A parameter list is used to pass information between CICS and the data table exit programs.

In the CICSTS52.CICS.SDFHMAC library, CICS supplies a copybook named DFHXDTDS that contains a DSECT to define this parameter list. Include a COPY DFHXDTDS statement in each of your exit programs. The DSECT is shown in Figure 22 on page 398.

The field names used in this DSECT are referenced in the user exit descriptions that follow the figure.

```
*********************************************************************
*                                                                   *
*   Data Table Parameter List for User Exits XDTRD, XDTAD and XDTLC.  *
*                                                                   *
*   Some of the parameters are only used by one or two of the exits.  *
*   This is indicated in the comments for those parameters.         *
*   The comments also indicate whether the field is used for input    *
*   (In), output (Out), or both (In/Out).                           *
*                                                                   *
*   This definition can be used by exit programs running on CICS      *
*   regions which are at a level to support coupling facility data    *
*   tables (CFDT), providing the UEPDTCFT flag is used to test        *
*   whether the exit has been invoked from within coupling facility   *
*   data tables support, and that parameters which are specific to    *
*   CFDT support are only used when it is set.  CFDT support will     *
*   only be available to exit programs running on CICS regions at     *
*   the CICS Transaction Server version 1 release 3 level or higher.  *
*                                                                   *
*   This definition can be used by exit programs running on CICS      *
*   regions which are at a level to support shared data tables (SDT), *
*   or which have SDT support installed, providing the UEPDTSDT flag  *
*   is used to test whether the exit has been invoked from within     *
*   shared data tables support, and that the parameters which are     *
*   specific to SDT support are only used when it is set.  SDT        *
*   support will only be available to CICS regions running at the     *
*   CICS/ESA version 4 release 1 level or higher (or running on       *
*   CICS/ESA version 3 release 3 if the Shared Data Tables feature    *
*   is installed).                                                  *
*                                                                   *
*   This definition can also be used by exit programs running on      *
*   CICS regions which are not at a level to support either SDT or    *
*   CFDT, indicated by both the UEPDTCFT and UEPDTSDT flags being     *
*   off.  In this case, only the parameters which relate to the       *
*   basic data tables support can be used.  Basic data tables         *
*   support will only be available to CICS regions running on one     *
*   of the following levels:                                        *
*   - CICS/MVS version 2 (plus data tables SPE on some releases)      *
*   - CICS/ESA version 3 releases 1 or 2                            *
*   - CICS/ESA version 3 release 3 if SDT feature is NOT installed    *
*                                                                   *
*   Careful use of these flags, and of the parameters which relate    *
*   to the various kinds of data tables support, should allow the     *
*   same user exit program to be used for more than one kind of data  *
*   table.                                                          *
*                                                                   *
*********************************************************************
DT_UE_PLIST_DSECT  DSECT ,
DT_UE_PLIST        DS  0XL84           Data Table User Exits          X
                                       Parameter List
UEPDTNAM             DS  CL8           Data table name (In)
UEPDTFLG             DS  0CL1          Flags (In):
*-------------------------------------------------------------------*
*            The UEPDTSDT and UEPDTCFT flags indicate whether the   *
*            exit has been invoked for shared data tables or for    *
*            coupling facility data tables support.  If neither is  *
*            set, then the exit has been invoked under the basic    *
*            data tables support which pre-dated shared data tables.*
*                                                                   *
*            The UEPDTCMT and UEPDTUMT flags are available only to  *
*            exits which have been invoked by shared data tables    *
*            support.  They distinguish the two kinds of shared     *
*            data table.  Please note that on releases earlier than *
*            CICS Transaction Server 1.3, the UEPDTUMT flag is NOT  *
*            available; on these releases, a user-maintained data   *
*            table is implied by the UEPDTCMT flag being turned off.*
*                                                                   *
*            If the exit has been invoked by CFDT support, then the *
*            data table can only be a coupling facility data table, *
*            so there are no extra flags to identify the kind of    *
*            data table when the exit has been invoked by coupling  *
*            facility data tables support.                          *
*                                                                   *
*            The UEPDTOPT flag is available to exits which have     *
```

The user exits should set a return code in register 15. The return code values are supplied by the DFHUEXIT macro. The valid values for each user exit are given in the following descriptions.

If you want your exit programs still to work for basic data tables as well as for shared data tables, you can check UEPDTFLG to find out which version of data tables support invoked the exit program. For shared data tables, this flag byte also indicates which type of data table is being used and whether the exit program is being invoked during loading.

The exit program should use either the filename (field UEPDTNAM) or the name of the source data set (see fields UEPDTDSN and UEPDTDSL) to determine whether any action is to be taken for this file.

You can enable several exit programs at the same exit point, each of which, for example, takes action for a particular file or data set.

XDTRD user exit

The XDTRD user exit is invoked just before CICS attempts to add a record that has been retrieved from the source data set to the data table. You can choose whether to load the record into the data table or not. For a user-maintained data table, you can also modify the record.

XDTRD is normally invoked when the loading process retrieves a record during the sequential copying of the source data set. However, it can also be invoked when an application retrieves a record that is not in the data table and one of the following conditions applies:

- For a user-maintained data table, loading is still in progress.
- For a CICS-maintained data table, loading terminated before the end of the source data set was reached (because, for example, the data table was full).

The record retrieved from the source data set is passed as a parameter to the user exit program—see fields UEPDTRA and UEPDTRL. This program can choose (depending, for example, on the key value—see fields UEPDTKA and UEPDTKL) whether to include the record in the data table or not.

Alternatively, the exit program can request that all subsequent records up to a specified key are skipped—see field UEPDTSKA; these records are not passed to the exit program. This facility is available only during loading. You can specify the key as a complete key, or you can specify just the leading characters by padding the skip-key area with binary zeros.

The action required is indicated by setting the return code. Depending on the return code value, the following action is taken by CICS:

Table 18. Return codes for XDTRD user exit. A value of UERCPURG should be returned if the exit program has received a PURGED response to a call that it has issued.

Return code	Action
UERCDTAC	Include the record in the data table. This is the default if the exit is not activated.
UERCDTRJ	Do not include the record in the data table.
UERCDTOP	Skip over this record and the following records until a key is found that is equal to or greater than the key specified in the skip-key area.

For a user-maintained data table, the program can also modify the data in the record to reduce the storage needed for the data table. Application programs that use the data table must be aware of any changes made to the record format by the exit program. If the record length is changed, the exit program must set the new length in the parameter list—see field UEPDTRL. The new length must not exceed the data buffer length—see field UEPDTRBL.

Sample XDTRD exit program

This sample program demonstrates the use of the XDTRD user exit for shared data tables.

The sample program is provided in the SDFHSAMP library. Copybook DFHXDTDS defines the data tables user exit parameter list that is used in the sample. DFHXDTDS is shown in "Communicating between CICS and shared data table exit programs" on page 397.

```
        TITLE 'DFH$DTRD - Sample XDTRD Global User Exit Program'
*****************************************************************
*                                                               *
*   MODULE NAME = DFH$DTRD                                      *
*                                                               *
* DESCRIPTIVE NAME = %PRODUCT Data Tables Sample XDTRD Exit     *
*                                                               *
* STATUS = %JUP0                                                *
*                                                               *
* FUNCTION =                                                    *
*        Sample Global User Exit Program to run at the XDTRD exit *
*                                                               *
*   The program selects records for inclusion in a data table.  *
*                                                               *
* ------------------------------------------------------------- *
*   NOTE that this program is only intended to DEMONSTRATE the use *
*   of the data tables user exit XDTRD, and to show the sort of *
*   information which can be obtained from the exit parameter list. *
* IT SHOULD BE TAILORED BEFORE BEING USED IN A PRODUCTION ENVIRONMENT *
* ------------------------------------------------------------- *
*                                                               *
*   This global user exit program will be invoked, if enabled, when *
*   a record which has been fetched from the source data set is about *
*   to be added to the data table.                              *
*                                                               *
*   The program is intended for use in the following situations : *
*                                                               *
*   (1) on CICS systems that have only the original data table  *
*       support (ie.before shared data tables)                  *
*   (2) on CICS systems that have shared data table support     *
*   (3) on CICS systems that have shared date table support AND *
*       coupling facility data table support                    *
*                                                               *
*   Flags are passed in the data tables parameter list which may *
*   be used to determine whether the exit has been invoked from *
*   a system which supports shared data tables or coupling facility *
*   data tables.                                                *
*                                                               *
*   The purpose of the program is to demonstrate the use of the *
*   option to optimise the data table load by skipping over ranges of *
*   key values which are to be excluded from the table.  This option *
*   is only allowed for shared data tables and coupling facility *
*   data tables but the program also illustrates that           *
*   individual records can be rejected when the exit            *
*   is not invoked by the data table loading transaction, or when *
*   shared data tables support or coupling facility data table  *
*   support are not in use.                                     *
*                                                               *
*   If the program has been invoked by shared data tables support *
*   or coupling facility data table support then it checks      *
*   whether the source data set name passed to it is the one    *
*   defined by the constant EXITDSN.  If so, and the exit has   *
*   been invoked from the loading transaction, then the skip-during- *
*   load option will be used to skip (not attempt to load) any  *
*   records except those whose keys start with the two characters in *
*   in EXITKEY.                                                 *
*                                                               *
*   If the program has not been invoked by shared data tables,  *
*   or coupling facility data tables, it uses the data          *
*   table name rather than the source DSname to check           *
*   whether this is the file from which records are to be rejected. *
*   If the table name matches the constant EXITFILE then only records *
*   whose keys start with the two characters in EXITKEY will be *
*   accepted for inclusion in the table.                        *
*                                                               *
*   A number of the actions taken are for illustrative purposes only, *
*   rather than being the recommended way in which to code an XDTRD *
*   exit program - for example, the program demonstrates how the *
*   keylength passed to the exit can be used to avoid having to know *
*   the keylength of the source data set, whereas in practice this *
*   might well be known;  and the program chooses to reject any *
*   records which are not presented to it by the loading transaction, *
*   whereas it would be more realistic to accept all records in the *
```

XDTAD user exit

XDTAD is invoked for each record that is added to the source data set after initial loading. You can choose whether to add the record to the data table or not. This user exit cannot modify the records because, as the records are written by the application, it is assumed that they are already in the format used in the data table.

The XDTAD user exit is invoked when a write request is issued to a data table.

- For a user-maintained data table, the user exit is invoked once—before the record is added to the data table.
- For a CICS-maintained data table, the user exit is invoked twice—before the record is added to the source data set and then again before the record is added to the data table.

The record written by the application is passed as a parameter to the user exit program—see fields UEPDTRA and UEPDTRL. This program can choose (depending on the key value, for example—see fields UEPDTKA and UEPDTKL) whether to include the record in the data table or not. This decision is indicated by setting the return code.

Depending on the return code value, the following action is taken by CICS:

Table 19. Return codes for XDTAD user exit. A value of UERCPURG should be returned if the exit program has received a PURGED response to a call that it has issued.

Return code	Action
UERCDTAC	Add the record to the data table. This is the default if the exit is not activated.
UERCDTRJ	Do not add the record to the data table.

The XDTAD exit must not modify the data in the record. If you used XDTRD to truncate the data records when the user-maintained data table was loaded, you must code your application so that it only tries to write records of the correct format for the data table.

Sample XDTAD exit program

This sample program demonstrates the use of the XDTAD user exit for shared data tables.

The sample program is provided in the SDFHSAMP library. Copybook DFHXDTDS defines the data tables user exit parameter list that is used in the sample. DFHXDTDS is shown in "Communicating between CICS and shared data table exit programs" on page 397.

```
        TITLE 'DFH$DTAD - Sample XDTAD Global User Exit Program'
*****************************************************************
*                                                               *
*   MODULE NAME = DFH$DTAD                                      *
*                                                               *
* DESCRIPTIVE NAME = %PRODUCT Data Tables Sample XDTAD Exit     *
*                                                               *
* STATUS = %JUP0                                                *
*                                                               *
* FUNCTION =                                                    *
*        Sample Global User Exit Program to run at the XDTAD exit *
*                                                               *
*   The program selects records for inclusion in a data table.  *
*                                                               *
* ------------------------------------------------------------- *
*   NOTE that this program is only intended to DEMONSTRATE the use *
*   of the data tables user exit XDTAD, and to show the sort of *
*   information which can be obtained from the exit parameter list. *
* IT SHOULD BE TAILORED BEFORE BEING USED IN A PRODUCTION ENVIRONMENT *
* ------------------------------------------------------------- *
*                                                               *
*   This global user exit program will be invoked, if enabled, when *
*   a WRITE request is issued to a data table.                  *
*                                                               *
*   The program is intended for use in the following situations : *
*                                                               *
*   (1) on CICS systems that have only the original data table  *
*       support (ie.before shared data tables)                  *
*   (2) on CICS systems that have shared data table support     *
*   (3) on CICS systems that have shared date table support AND *
*       coupling facility data table support                    *
*                                                               *
*   Flags are passed in the data tables parameter list which may *
*   be used to determine whether the exit has been invoked from *
*   a system which supports shared data tables or coupling facility *
*   data tables.                                                *
*                                                               *
*   The purpose of the program is to demonstrate the use of the XDTAD *
*   global user exit to select only certain records for inclusion in *
*   a data table.  In this example, the selection is made on the *
*   basis of key values.  If shared data tables support or      *
*   coupling facility data tables support is being used         *
*   and the source data set for the data table is that          *
*   specified by the constant EXITDSN, then the program will select *
*   particular keys for inclusion in the data table, and reject *
*   others.  For all other source data sets, or if shared data table *
*   support or coupling facility data table support is not in   *
*   in use, then all records will be accepted.                  *
*                                                               *
*   The record selection is made on the basis of the value of the 6th *
*   character in the record key.  This is for illustrative purposes *
*   only, as it is unlikely to be the criterion for selection in a *
*   realistic environment.  For example, for a shared CICS-maintained *
*   data table, it might be desirable to select a group of records *
*   which are known to be very frequently read by applications  *
*   running in other CICS regions in the MVS system.            *
*                                                               *
*   The trace flag passed to the exit is set ON if File Control (FC) *
*   level 1 tracing is enabled.                                 *
*                                                               *
* NOTES :                                                       *
*    DEPENDENCIES = S/390                                       *
*          DFH$DTAD, or an exit program which is based on this  *
*          sample, must be defined on the CSD as a program      *
*          (with DATALOCATION(ANY)).                            *
*    RESTRICTIONS =                                             *
*          This program is designed to run on CICS/ESA 3.3 or later *
*          release.  It requires the DFHXDTDS copybook to be    *
*          available at assembly time.                          *
*    REGISTER CONVENTIONS = see code                            *
*    MODULE TYPE = Executable                                   *
*    PROCESSOR = Assembler                                      *
*    ATTRIBUTES = Read only, AMODE 31, RMODE ANY                *
```

XDTLC user exit

The XDTLC user exit is invoked when the loading of the data table is complete, whether successful or not. The user exit is not invoked if the data table is closed for any reason before loading is complete.

The exit program is informed if the loading did not complete successfully; see field UEPDTORC. This could occur, for example, if the maximum number of records was reached or there was insufficient virtual storage. In this case, the exit program can request that the file is closed immediately, by setting the return code.

Depending on the return code value, the following action is taken by CICS:

Table 20. Return codes for XDTLC user exit. A value of UERCPURG should be returned if the exit program has received a PURGED response to a call that it has issued.

Return code	Action
UERCDTOK	No action; the file remains open. This is the default if the exit is not activated.
UERCDTCL	Close the file.

Sample XDTLC exit program

This sample program demonstrates the use of the XDTLC user exit for shared data tables.

The sample program is provided in the SDFHSAMP library. Copybook DFHXDTDS defines the data tables user exit parameter list that is used in the sample. DFHXDTDS is shown in "Communicating between CICS and shared data table exit programs" on page 397.

```
         TITLE 'DFH$DTLC - Sample XDTLC Global User Exit Program'
*****************************************************************
*                                                               *
*   MODULE NAME = DFH$DTLC                                      *
*                                                               *
* DESCRIPTIVE NAME = %PRODUCT Data Tables Sample XDTLC Exit     *
*                                                               *
* STATUS = %JUP0                                                *
*                                                               *
* FUNCTION =                                                    *
*        Sample Global User Exit Program to run at the XDTLC exit *
*                                                               *
*   The program rejects a data table if its load did not complete OK. *
*                                                               *
* ------------------------------------------------------------- *
*   NOTE that this program is only intended to DEMONSTRATE the use *
*   of the data tables user exit XDTLC, and to show the sort of *
*   information which can be obtained from the exit parameter list. *
* IT SHOULD BE TAILORED BEFORE BEING USED IN A PRODUCTION ENVIRONMENT *
*   If this program is modified to accept a load that has failed *
*   or reject a load that has been successful, it is the        *
*   responsibility of the program to issue a message indicating *
*   what has happened. Any message output by CICS File Control will *
*   only reflect what happened before modification of return codes *
*   by the exit program.                                        *
* ------------------------------------------------------------- *
*                                                               *
*   This global user exit program will be invoked, if enabled, when *
*   the load of a data table has completed.                     *
*                                                               *
*   The program is intended for use in the following situations : *
*                                                               *
*   (1) on CICS systems that have only the original data table  *
*       support (ie.before shared data tables)                  *
*   (2) on CICS systems that have shared data table support     *
*   (3) on CICS systems that have shared date table support AND *
*       coupling facility data table support                    *
*                                                               *
*   Flags are passed in the data tables parameter list which may *
*   be used to determine whether the exit has been invoked from *
*   a system which supports shared data tables or coupling facility *
*   data tables.                                                *
*                                                               *
*   The program will issue a user trace entry if tracing is enabled, *
*   then check the setting of the load completion indicator.    *
*   If this shows that loading failed to complete successfully, then *
*   the exit program will set a return code that rejects the table *
*   by requesting that it be closed.                            *
*                                                               *
*   The trace flag passed to the exit is set ON if File Control (FC) *
*   level 1 tracing is enabled.                                 *
*                                                               *
* NOTES :                                                       *
*    DEPENDENCIES = S/390                                       *
*          DFH$DTLC, or an exit program which is based on this  *
*          sample, must be defined on the CSD as a program      *
*          (with DATALOCATION(ANY)).                            *
*    RESTRICTIONS =                                             *
*          This program is designed to run on CICS/ESA 3.3 or later *
*          release.  It requires the DFHXDTDS copybook to be    *
*          available at assembly time.                          *
*    REGISTER CONVENTIONS = see code                            *
*    MODULE TYPE = Executable                                   *
*    PROCESSOR = Assembler                                      *
*    ATTRIBUTES = Read only, AMODE 31, RMODE ANY                *
*                                                               *
*---------------------------------------------------------------*
*                                                               *
* ENTRY POINT = DFH$DTLC                                        *
*                                                               *
*     PURPOSE =                                                 *
*         Described above                                       *
*                                                               *
```

Activating user exits for data tables

To activate the data table user exits, complete these steps.

1. Decide which user exits you want to use. For a description of each user exit, see Chapter 4, "Customizing data tables using user exits," on page 397.
2. Write the user exit programs. Examples are included in Chapter 4, "Customizing data tables using user exits," on page 397.
3. Define the user exit programs to CICS, using the CEDA DEFINE PROGRAM command as described in PROGRAM resources in Reference -> System definition.
4. Activate the user exits, using the EXEC CICS ENABLE command as described in the *CICS System Programming Reference*. If required, you can later deactivate the user exits using the EXEC CICS DISABLE command.

Unless you control the opening of a data table explicitly, with a CEMT or EXEC CICS command, you should probably activate the user exits during CICS startup. Otherwise the loading of the data table might begin before the user exits are activated. To activate the user exits during startup, you need to:

1. Write one or more program list table postinitialization (PLTPI) programs that include the EXEC CICS ENABLE commands to activate the user exits (for programming information about PLTPI programs, see the *CICS Customization Guide*).
2. Define a program list table (PLT) with an entry for each of those PLTPI programs, as described in the *CICS Resource Definition Guide*.
3. Specify the **PLTPI=suffix** parameter for system initialization, as described in the *CICS System Definition Guide*. Use the suffix of the PLT that was defined in the previous step. This causes the PLTPI programs to be executed in the second stage of initialization, before any files are opened.

You can use PLT shutdown (PLTSD) programs in a similar way to disable the user exits during CICS shutdown.

Part 2. Customizing with initialization and shutdown programs

Chapter 5. Writing initialization and shutdown programs

You can write programs to run during the initialization and shutdown phases of CICS processing. Any program that is to run at these times must be defined to CICS in a program list table (PLT).

For information about how to code the PLT, see Program list table (PLT) in Reference -> System definition.

Writing initialization programs

Any program that is to run during CICS initialization must be specified in a program list table (PLT), and the suffix of that PLT must be named on the program list table post initialization (PLTPI) system initialization parameter.

There are two phases of program list table (PLT) execution, separated by the DFHDELIM statement in the PLT.

First phase PLT programs

During the early stages of CICS initialization processing, the only PLT programs that can run are those that contain the enabling commands for global and task-related user exit programs. These programs are specified in the first part of the PLTPI list (before the DFHDELIM statement), so you can enable exit programs that are needed during recovery.

First phase PLT programs are run during the second stage of CICS initialization, which is the phase that is returned as SECONDINIT by the **EXEC CICS INQUIRE SYSTEM** command or the **XPI INQUIRE_SYSTEM** call.

Dynamic LIBRARY resources are installed, or restored and reactivated, after first stage PLT programs run, but before second stage PLT programs run. First stage PLT programs must be included in data sets in DFHRPL, but second stage PLT programs can be included in, and loaded from, dynamic LIBRARY resources.

The following points apply to all first phase PLTPI programs:

- The programs must be written in assembler language.
- They must run AMODE 31 or AMODE 64.
- The only EXEC CICS commands that they can contain are as follows:
  - **ASSIGN APPLID**
  - **ASSIGN INITPARM**
  - **ENABLE**
  - **EXTRACT EXIT**

  Because this stage occurs before recovery when initialization is incomplete, no other CICS services can be called.
- If a first phase PLTPI program enables an exit program that issues any of the XPI calls INQUIRE_APP_CONTEXT, INQUIRE_MONITORING_DATA, MONITOR, TRANSACTION_DUMP, or WRITE_JOURNAL_DATA, it must not specify the START option on the **EXEC CICS ENABLE COMMAND**.
- First phase PLTPI programs must not enable any task-related user exit program with the TASKSTART option.

- Because first phase PLT programs run so early in CICS initialization, no resource definitions are available. You cannot use installed PROGRAM definitions (or the program autoinstall user program) to define first phase PLT programs to CICS, or define the user exit programs that are enabled by first phase PLT programs. Instead, CICS installs default definitions automatically. Even if program autoinstall is specified as active on the **PGAIPGM** system initialization parameter, the autoinstall user program is not called to allow the definitions to be modified.

  This type of autoinstall by CICS is known as *system autoinstall*.

  CICS defines first phase PLT programs, and the user exit programs that they enable, with the following attributes:

  ```
  LANGUAGE(Assembler)
  RELOAD(No)
  STATUS(Enabled)
  CEDF(No)
  DATALOCATION(Below)
  EXECKEY(CICS)
  EXECUTIONSET(Fullapi)
  CONCURRENCY(Quasirent)
  ```

  Always write global user exit programs to be threadsafe. However, the system-autoinstalled program definition specifies CONCURRENCY(Quasirent); that is, the exit programs are defined as quasi-reentrant. To define a first phase PLT global user exit program as threadsafe, specify the THREADSAFE keyword on the **EXEC CICS ENABLE** command. This value overrides the CONCURRENCY(QUASIRENT) setting on the system-autoinstalled program definition.
- You cannot use Debug Tool to debug a first phase PLT program.

Second phase PLT programs

During the final stages of CICS initialization, most CICS services are available to PLT programs. These programs are specified in the second part of the PLTPI list (after the DFHDELIM entry).

Second phase PLT programs are run during the third stage of CICS initialization, which is the phase that is returned as `THIRDINIT` by the **EXEC CICS INQUIRE SYSTEM** command or the **XPI INQUIRE_SYSTEM** call.

The limitations on the services that are available to second phase PLTPI programs are as follows:

- Because interregion communication (IRC) and intersystem communication (ISC) have pseudo-terminal entries associated with their function, you cannot run any IRC or ISC functions during PLTPI processing, including ISC over SNA and IP interconnectivity (IPIC). Second phase PLT programs must not issue any EXEC CICS commands, even INQUIRE commands, related to transaction routing (and therefore pseudo-terminals) that attempt to access remote resources.

  This restriction occurs if the remote resource is not available. The remote resource might be unavailable for one of the following reasons:
  - AUTOCONNECT=NO is specified on the connection definition.
  - The remote region is not running.
  - The remote resource in the remote region is not available.
  - The link is broken; because of a network problem, for example.

  However, if the connection with the remote region is available and the resource in the remote region is also available, this restriction does not apply.

  Note: A pseudo-terminal:

  - Must be a surrogate TCTTE that exists only in an AOR
  - Can be used only in a transaction routing environment
  - Cannot exist with distributed program link (DPL) requests
  - Cannot exist with any type of function shipping request
  - Cannot exist in a distributed transaction.
- PLTPI programs can request services that could suspend the issuing task, but suspending the task can affect the time at which control is given to CICS. The suspension must not require the decision to resume to be taken by another task.
- Although PLTPI programs can issue interval control **START** commands, the requested transactions are not attached before the initialization stages have completed, unless the ATTACH option is specified. **START ATTACH** allows a **START** command that is issued in a PLTPI program to take effect before initialization has completed. If you use **START** without the ATTACH option, the invoked transaction does not start until after the PLTPI programs have completed.
- PLTPI programs must not issue memory dump requests.
- PLTPI programs must not use the **EXEC CICS PERFORM SHUTDOWN** command. If the PLTPI uses this command, a severe error occurs in DFHDMDM. The **EXEC CICS PERFORM SHUTDOWN IMMEDIATE** command is allowed.
- PLTPI programs must not be Java programs that run in a JVM server, because JVM servers start asynchronously to PLTPI programs.
- Second stage initialization and second stage quiesce PLT programs do not require program resource definitions. If they are not defined, they are autoinstalled on the system (irrespective of the program autoinstall system initialization parameters). The autoinstall exit is not called to allow the definition to be modified. The programs are defined with the following attributes:

  ```
  LANGUAGE(ASSEMBLER)
  STATUS(ENABLED)
  CEDF(NO)
  DATALOCATION(BELOW)
  EXECKEY(CICS)
  EXECUTIONSET(FULLAPI)
  ```

  As a result, system-autoinstalled programs have a default CONCURRENCY setting of QUASIRENT, and a default API setting of CICSAPI. To run PLT programs with different CONCURRENCY or API settings, or for C or C++ programs that are compiled with the XPLINK compiler option, provide an appropriate resource definition. Alternatively, for Language Environment conforming programs, use the CICSVAR runtime option to set the appropriate CONCURRENCY and API values. See the *CICS Application Programming Guide*
- You cannot use Debug Tool to debug a second phase PLT program.

Effect of delayed recovery on PLTPI processing

Because recovery processing does not take place until PLTPI processing is complete, PLT programs may fail during an emergency restart if they attempt to access resources protected by retained locks. If PLT programs are not written to handle the LOCKED exception condition, they abend with an AEX8 abend code.

If successful completion of PLTPI processing is essential before your CICS applications are allowed to start, consider alternative methods of completing necessary PLT processing. You may have to allow emergency restart recovery processing to finish, and then complete the failed PLTPI processing when the locks have been released.

Writing shutdown programs

Any program that is to run during CICS shutdown must be defined in a program list table (PLT), and the PLT must be named on the program list table shutdown (PLTSD) system initialization parameter.

You can override the PLTSD value by using the **Shutdown** option from the CICS Explorer **Regions** operations view or by providing a PLT name on the **CEMT PERFORM SHUTDOWN** command, or on the **EXEC CICS PERFORM SHUTDOWN** command. If a PLTSD program abends, sync point rollback occurs.

First quiesce phase PLT programs

Programs that are to execute during the first quiesce stage of CICS shutdown are specified in the first half of the PLT (before the DFHDELIM statement).

You must define first stage PLTSD programs to CICS. You can either define the programs statically, or use program autoinstall. You cannot define Java programs that run in a JVM server.

Although terminals are still available during the first quiesce stage, tasks that are started by terminal input are rejected unless they are named in a shutdown transaction list table (XLT), or are CICS-supplied transactions, such as CEMT, CSAC, CSTE, and CSNE, that are defined as SHUTDOWN(ENABLED) in the supplied definitions.

The first quiesce stage is complete when all of the first-stage PLT programs have executed, and when there are no user tasks in the system.

You cannot use Debug Tool to debug a PLT program during the first quiesce stage.

PLT programs for the second quiesce stage

Programs that are to execute during the second quiesce stage of CICS shutdown are specified in the second half of the PLT (after the DFHDELIM statement).

Second stage initialization and second stage quiesce PLT programs do not require program resource definitions. If they are not defined, they are system autoinstalled (irrespective of the program autoinstall system initialization parameters). This means that the autoinstall exit is not called to allow the definition to be modified. The programs are defined with the following attributes:

```
LANGUAGE(ASSEMBLER)   STATUS(ENABLED)   CEDF(NO)
DATALOCATION(BELOW)   EXECKEY(CICS)
EXECUTIONSET(FULLAPI)
```

As a result, system autoinstalled programs have a default CONCURRENCY setting of QUASIRENT, and a default API setting of CICSAPI.

- For those threadsafe PLT programs that are defined with the OPENAPI value for the API attribute, or are C or C++ programs compiled with the XPLINK compiler option, provide an appropriate resource definition. Alternatively, for Language Environment conforming programs, use the `CICSVAR` runtime option to set the appropriate CONCURRENCY and API values. See Defining runtime options for Language EnvironmentDefining runtime options for Language Environment.

During the second quiesce stage, no new tasks can start, and no terminals are available. Because of this, second phase PLT programs must not cause other tasks to be started, and they cannot communicate with terminals. Further, second phase PLT programs must not cause any resource security checking or DB2 calls to be performed. A PLT program cannot be a Java program that runs in a JVM server.

If a transaction abend occurs while the PLTSD program is running, CICS remains in a permanent wait state. To avoid this happening, ensure that your PLTSD program handles **all** abend conditions.

The second quiesce stage is complete when all of the second phase PLT programs have been executed.

You cannot use Debug Tool to debug a PLT program during the second queisce stage.

The shutdown assist utility program, DFHCESD

CICS provides a shutdown assist transaction, that can be run during the first quiesce stage of shutdown. It can be run on a normal or an immediate shutdown.

You specify the name of the shutdown transaction on the **SDTRAN** system initialization parameter, or on the SDTRAN option of the **PERFORM SHUTDOWN** and **PERFORM SHUTDOWN IMMEDIATE** commands. You can also specify that no shutdown assist transaction is to be run. If you do specify that no shutdown assist transaction is to be run, the following processing occurs:

- On a normal shutdown, CICS waits for all running tasks to finish before entering the second stage of quiesce. Long running or conversational transactions can cause an unacceptable delay or require operator intervention.
- On an immediate shutdown, CICS does not allow running tasks to finish and backout is not performed until emergency restart. This can cause an unacceptable number of units of work to be shunted, and locks to be retained unnecessarily.

The purpose of the shutdown assist transaction is to help solve these problems; that is, to ensure that as many tasks as possible commit or back out cleanly within a reasonable time.

The default shutdown assist transaction is CESD, which starts the CICS-supplied program DFHCESD. DFHCESD attempts to purge and back out long-running tasks using increasingly stronger techniques. It ensures that as many tasks as possible commit or back out cleanly, enabling CICS to shut down in a controlled manner. For information about DFHCESD, and about how to write your own shutdown assist transaction, see Shutdown assist program (DFHCESD) in Reference -> Utilities.

General considerations when writing initialization and shutdown programs

If you are writing initialization and shutdown programs, consider how this affects your PLT, PLTPI, and PLTSD programs.

The following information applies to both initialization and shutdown programs:

- Terminate all PLT programs with an **EXEC CICS RETURN** command.

- PLT programs receive control in primary-space translation mode. For information about translation modes, see the *z/Architecture Principles of Operation* manual. PLT programs must return control to CICS in the same mode, and must restore any general-purpose registers or access registers that they use.
- All PLTPI programs run under the CICS internal transaction name, CPLT. Therefore, because CICS internal transactions are defined with the WAIT indoubt attribute set to YES, an indoubt failure that occurs while a PLTPI program is running causes the relevant UOW to be shunted. The PLTPI program abends ASP1, and CICS runs the next program defined in the PLTPI table, if any.
- PLTSD programs run under the transaction that issued the **PERFORM SHUTDOWN** command. The CEMT transaction is defined with WAIT(YES). Therefore, if shutdown is as the result of a **CEMT PERFORM SHUTDOWN** command or the **Shutdown** option from the CICS Explorer **Regions** operations view, an indoubt failure that occurs while a PLTSD program is running causes the UOW to be shunted. If, however, shutdown is as the result of a user transaction issuing an **EXEC CICS PERFORM SHUTDOWN** command, whether an indoubt failure causes the UOW to be shunted or a forced decision taken depends on the indoubt attributes of the user transaction. For details of the indoubt options of the **CEDA DEFINE TRANSACTION** command, see TRANSACTION attributes in Reference -> System definition.
- The TRANSACTION resource definition for CEMT specifies TASKDATALOC(ANY). The CEMT transaction therefore uses 31-bit storage above the 16 MB line. If you use CEMT to shut down CICS and have PLTSD programs that are AMODE(24), an AEZC abend will occur. To avoid this situation, modify the shutdown program so that it is AMODE(31) and update the appropriate program definition.

Storage keys for PLT programs

You need to consider the following (whether or not you are running CICS with the storage protection facility):

- The execution key in which your PLT programs are invoked
- The storage key of data storage obtained for your PLT programs.

Execution key for PLT programs

CICS always gives control to PLT programs in CICS key.

Even if you specify EXECKEY(USER) on the program resource definition, CICS forces CICS key when it passes control to any PLT programs invoked during initialization or shutdown. However, if a PLT-defined *shutdown* program itself passes control to another program (via a link or transfer-control command), the program thus invoked executes according to the execution key (EXECKEY) defined in its program resource definition.

Important: You are strongly recommended to specify EXECKEY(CICS) when defining both PLT programs and programs to which a PLT program passes control.

Data storage key for PLT programs

The content of the data storage key used by PLT programs depends on how the storage is obtained.

Storage can be obtained in the following ways:

- Any working storage requested by the PLT program is in the key set by the TASKDATAKEY value of the transaction under which the PLT program is started. If PLT programs run during initialization (PLTPI programs), the

transaction is always an internal CICS transaction, in which case the TASKDATAKEY value is always CICS. For programs that run during shutdown (PLTSD programs), the setting depends on the transaction you use to issue the shutdown command. If you select the **Shutdown** option from the CICS Explorer **Regions** operations view or issue the **CEMT PERFORM SHUTDOWN** command, the TASKDATAKEY value is always CICS. If you run a user-defined transaction, to start a program that issues an **EXEC CICS PERFORM SHUTDOWN** command, the TASKDATAKEY can be either USER or CICS.

- PLT programs can use **EXEC CICS** commands to obtain storage by issuing:
  - Explicit **EXEC CICS GETMAIN** commands
  - Implicit storage requests as a result of EXEC CICS commands that use the SET option

  The default storage key for storage obtained by **EXEC CICS** commands is set by the TASKDATAKEY value of the transaction under which the PLT program is started, exactly as described for working storage.

  As an example, consider a transaction defined with TASKDATAKEY(USER) that causes a PLT shutdown program to be started. In this case, any implicit or explicit storage acquired by the PLT program with an **EXEC CICS** command is, by default, in user-key storage. However, on an EXEC CICS GETMAIN command, the PLT program can override the TASKDATAKEY option by specifying either CICSDATAKEY or USERDATAKEY.

Part 3. Customizing with user-replaceable programs

A user-replaceable program is a CICS-supplied program that is always invoked at a particular point in CICS processing, as if it were part of the CICS code. You can modify the supplied program by including your own logic, or replace it with a version that you write yourself.

When creating your own versions of user-replaceable programs, you must follow this guidance:

- You can code user-replaceable programs in any of the languages supported by CICS (that is, in assembler language, COBOL, PL/I, or C). An assembler-language version of most programs is provided, in source form, in the CICSTS52.CICS.SDFHSAMP library. COBOL, PL/I, or C versions are provided for some programs. The description of each program lists the sample programs, copy books, and macros supplied in each case.
- You can trap an abend in a user-replaceable program by making the program issue an **EXEC CICS HANDLE ABEND** command. However, if no **HANDLE ABEND** is issued, CICS does not abend the task but returns control to the CICS module that called the program. The action taken by the CICS module depends on the user-replaceable program concerned.
- Upon return from any user-replaceable program, CICS must always receive control in primary-space translation mode, with the original contents of all access registers restored, and with all general purpose registers restored (except for those which provide return codes or linkage information).

  For information about translation modes, refer to the *z/Architecture Principles of Operation*.
- In z/OS, do not install SVCs or PC routines that return control to their caller in any authorized mode: that is, in supervisor state, system PSW key, or APF-authorized. Doing so is contrary to the z/OS Statement of Integrity. If you invoke such services from CICS, you might compromise your system integrity, and any resultant problems will not be resolved by IBM Service.
- User-replaceable programs, and any programs invoked by user-replaceable programs, can be RMODE ANY but **must** be AMODE 31.
- You must ensure that user-replaceable programs are defined as local. User-replaceable programs cannot be run in a remote region. This rule applies to all user-replaceable programs, including the autoinstall control program and the dynamic routing program.
- User-replaceable programs produce only system dumps when a program check occurs; they do not produce transaction dumps.
- You can use the CICS Execution Diagnostic Facility (EDF) to test user-replaceable programs. However, EDF does not work if the initial transaction is a CICS-supplied transaction.

Chapter 6. Assembling and link-editing user-replaceable programs

Most of the user-replaceable programs are provided as command-level programs and must be translated, assembled and link-edited. CICS provides procedures to translate, assemble, and link-edit user-replaceable programs.

About this task

Except for DFHAPXPO, all programs are supplied as command-level programs, and must be translated before assembly and link-edit. You must code the translator options NOPROLOG and NOEPILOG with your versions of DFHZNEP, DFHTEP, and DFHXCURM.

Procedure

1. Copy the CICS-supplied user-replaceable program that you want to replace and edit the copy. The source for the CICS-supplied user-replaceable programs is installed in the CICSTS52.CICS.SDFHSAMP library. If the original SDFHSAMP is serviced, and a user-replaceable program is modified, you might want to reflect the changes in your own version of the code.
2. Translate, assemble, and link-edit your version of the program:
   - If you are replacing DFHAXPO, you do not have to translate the programs or link-edit the programs using the EXEC interface module. You can use the DFHASMVS procedure to compile these programs.
   - If you are replacing another program, use the appropriate CICS-supplied procedure to translate, assemble, and link-edit the program. For example, use the DFHEITAL procedure for AMODE(24) or AMODE(31) Assembler programs. You must link-edit the program with the EXEC interface module stub. This stub enables the program to communicate with the EXEC interface program, DFHEIP. The DFHEITAL procedure link-edits programs with the EXEC interface stub. If you use SMP/E, you can give the object-deck output after translation and assembly to SMP/E for link-editing.

     For information about using the procedures that are available for each language, see Using the CICS-supplied procedures to install programs in Deploying.

Example

The job stream in Figure 26 on page 420 is an example of the assembly and link-edit of a user-replaceable program. The figure is followed by some explanatory notes.

```
//ASSEMBLE EXEC DFHEITAL,
//      ASMBLR=ASMA90,
//      INDEX='CICSTS52.CICS',                              1
//      PROGLIB='your_loadlib',                             2
//      DSCTLIB='your_copylib',                             3
//      PARM.TRN='NOPROLOG,NOEPILOG',                       4
//      PARM.ASM='DECK,NOOBJECT,LIST,XREF(SHORT),RENT,ALIGN',
//      LNKPARM='LIST,XREF,RENT,MAP,AMODE(31),RMODE(ANY)'
//TRN.SYSIN DD DSN=your_sourcelib(program_name),DISP=SHR 5 6
//LKED.SYSIN DD *
 ENTRY program_name                                         7
 NAME program_name(R)
//*
```

Figure 26. Job stream to assemble and link-edit a user-replaceable program

Notes:

1 High-level qualifier of the CICS libraries.

2 The library into which the load module is link-edited.

3 Optionally, the name of a library containing your local Assembler macros and copy members.

4 These options are required for DFHXCURM, and for the supplied sample versions of DFHTEP and DFHZNEP.

5 `your_sourcelib` is the name of the library containing your modified version of the program.

6 `program_name` is the source member name of the user-replaceable program being assembled. The source member for the supplied DFHTEP sample is DFHXTEP. The source member for the supplied DFHZNEP sample is DFHZNEP0.

7 The input to the linkage-editor normally consists of the two statements shown here, with `program_name` replaced by the name of the user-replaceable program being compiled. There are some exceptions for some of the CICS-supplied sample programs, and these are shown in Figure 27.

Figure 27. Link-edit statements for DFHTEP and DFHZNEP

Link-edit statements for DFHTEP:

```
ENTRY DFHTEPNA
NAME DFHTEP(R)
```

Link-edit statements for DFHZNEP:

```
ENTRY DFHZNENA
NAME DFHZNEP(R)
```

Chapter 7. User-replaceable programs and the storage protection facility

When you are running CICS with the storage protection facility, you must decide the execution key in which your program runs and the storage key of data storage that is obtained by your program.

Execution key for user-replaceable programs

When you are running with storage protection active, CICS gives control to user-replaceable programs in CICS key.

Even if you specify EXECKEY(USER) on the PROGRAM resource, CICS forces CICS key when it calls the program. However, if a user-replaceable program itself passes control to another program, the called program runs according to the execution key (EXECKEY) defined in its PROGRAM resource.

Important: Specify EXECKEY(CICS) when defining both user-replaceable programs and programs to which a user-replaceable program passes control.

Data story key for user-replaceable programs

The storage key of storage used by user-replaceable programs depends on how the storage is obtained:

- The communication area passed to the user-replaceable program by its caller is always in CICS key.
- Any working storage obtained for the user-replaceable program is in the key set by the TASKDATAKEY of the transaction under which the program is started.
- User-replaceable programs can use EXEC CICS commands to obtain storage, by issuing:
  - Explicit **EXEC CICS GETMAIN** commands
  - Implicit storage requests as a result of EXEC CICS commands that use the SET option.

  The default storage key for storage obtained by EXEC CICS commands is set by the TASKDATAKEY of the transaction under which the user program is started.

  As an example, consider a transaction defined with TASKDATAKEY(USER) that causes a user-replaceable program to be called. In this case, any implicit or explicit storage acquired by the user program with an EXEC CICS command is, by default, in user-key storage. However, on an **EXEC CICS GETMAIN** command, the user program can override the TASKDATAKEY option by specifying either CICSDATAKEY or USERDATAKEY.

Chapter 8. Writing a program error program

You can write a program error program that is based on the supplied default program, DFHPEP.

The CICS-supplied default program error program (DFHPEP) contains code to obtain program addressability, access the communication area, and return control to CICS through an **EXEC CICS RETURN** command.

The source of DFHPEP is provided in assembler language and C versions; you can modify one of these to include your own logic, or you can write your own program error program in any language that is supported by CICS. The program error program is subject to specific restrictions:

- The name of your program must be DFHPEP.
- The program must not issue any EXEC CICS commands that use MRO or ISC facilities, such as distributed transaction processing or function shipping.
- The program must not issue any commands that access recoverable resources.
- The program cannot influence whether a transaction dump is taken.

The default DFHPEP module is a dummy module. To customize it, you must code the source yourself. A listing of DFHPEP is provided in Figure 28 on page 425. After you write your program error program, translate and assemble it, and use it to replace the supplied dummy program. For information about the job control statements necessary to assemble and link-edit user-replaceable programs, refer to Chapter 6, "Assembling and link-editing user-replaceable programs," on page 419.

Information available to DFHPEP in the communication area includes:

- The current abend code, at PEP_COM_CURRENT_ABEND_CODE.
- The original abend code, at PEP_COM_ORIGINAL_ABEND_CODE. The original and current abend codes are different if the transaction has experience more than one abend; for example, if the failing program abends while handling a previous abend. In this case, the original abend is the first abend that the transaction experienced.
- The EIB at the time of the last EXEC CICS command, at PEP_COM_USERS_EIB.
- The name of the program that suffered the (current) abend, at PEP_COM_ABPROGRAM. PEP_COM_ABPROGRAM identifies the program as follows:
  - If the abend occurred in a distributed program link (DPL) server program running in a remote system, it identifies the server program.
  - If the abend is a local ASRA, ASRB, or ASRD abend, it identifies the program in which the program check or operating system abend occurred.
  - In all other cases, it identifies the current program.
- The program status word (PSW) at the time of the (current) abend, at PEP_COM_PSW, or PEP_COM_PSW16 for a 16 byte PSW. The full contents of the PSW are significant only for the ASRA, ASRB, and ASRD abend codes. The last four bytes of PEP_COM_PSW and last eight bytes of PEP_COM_PSW16 (the PSW address) apply also to the AICA code.
- The GP registers (0-15) at the time of the (current) abend, at PEP_COM_REGISTERS.

- The execution key of the program at the time it suffered the (current) abend, at PEP_COM_KEY. The value of PEP_COM_KEY is significant only for the ASRA and ASRB abend codes.
- Whether the (current) abend occurred as the result of a storage protection exception, at PEP_COM_STORAGE_HIT. The value of PEP_COM_STORAGE_HIT is significant only for the ASRA abend code, and indicates which of the protected dynamic storage areas (the CDSA, RDSA, ECDSA, ERDSA, ETDSA, or GCDSA), if any, the failing program attempted to overwrite.
- Additional register information might be available. The additional information might include the 64-bit GP registers, the access registers, and the floating point registers. Indicators are set in the communication area to indicate which register values are available.
- If it is available, the Breaking Event Address is stored in the communication area. If it is not available, the Breaking Event Address is zero.
- Program status word interrupt information, at PEP_COM_INT.

Information about the PSW, registers, execution key, and type of protected storage the application attempted to overwrite is meaningful only if the abend occurred in the local system; these fields are set to zeros if the abend occurred in a DPL server program running in a remote system.

To disable the transaction, assign the value PEP_COM_RETURN_DISABLE to the PEP_COM_RETURN_CODE field. Otherwise, allow the field to default to zero, or set it to the value PEP_COM_RETURN_OK. CICS does not allow CICS-supplied transactions to be disabled; therefore do not attempt to disable transactions with IDs that start with the letter C.

Figure 28 on page 425 shows the assembler language source code of the default program error program.Figure 29 on page 426 shows the communication area.

```
DFHEISTG DSECT ,
*
*        Insert your own storage definitions here
*
         DFHPCOM TYPE=DSECT
*********************************************************************
* * * * *             P R O G R A M   E R R O R             * * * * *
* * * * *                   P R O G R A M                   * * * * *
*********************************************************************
DFHPEP   CSECT                     PROGRAM ERROR PROGRAM CSECT
DFHPEP   RMODE ANY
         DFHREGS ,                 EQUATE REGISTERS
         XR    R1,R1
         ICM   R1,B'0011',EIBCALEN Get Commarea length
         BZ    RETURNX             ...no Commarea; exit
         EXEC CICS ADDRESS COMMAREA(R2) ,
         USING DFHPEP_COMMAREA,R2
*
*        Insert your own code here
*
         LA    R1,PEP_COM_RETURN_OK
         B     RETURN
         DFHEJECT
*
RETURNER DS    0H                  Return for error cases
         LA    R1,PEP_COM_RETURN_DISABLE
RETURN   DS    0H
         ST    R1,PEP_COM_RETURN_CODE
RETURNX  DS    0H
         EXEC CICS RETURN ,
         END   DFHPEP
```

Figure 28. Source code of the default program error program (DFHPEP)

```
DFHPEP_COMMAREA DSECT
*
*                       Standard header section
*
PEP_COM_STANDARD              DS     0F
PEP_COM_FUNCTION              DS     CL1       Always '1'
PEP_COM_COMPONENT             DS     CL2       Always 'PC'
PEP_COM_RESERVED              DS     C         Reserved
*
*                       Abend codes and EIB
*
PEP_COM_CURRENT_ABEND_CODE    DS     CL4       Current abend code
PEP_COM_ORIGINAL_ABEND_CODE   DS     CL4       Original abend code
PEP_COM_USERS_EIB             DS     CL(EIBRLDBK-EIBTIME+L'EIBRLDBK)
*                                       EIB at last EXEC CICS command
*
*  Debugging information (program, PSW, registers and execution key at
*  time of abend, hit storage indicator).  If the abend occurred in a
*  DPL server program running remotely, only program is meaningful.
*
PEP_COM_DEBUG                 DS     0F
PEP_COM_ABPROGRAM             DS     CL8       Program causing abend
PEP_COM_PSW                   DS     CL8       PSW at abend
*                                              (codes ASRA, ASRB, AICA, ASRD)
PEP_COM_REGISTERS             DS     CL64      GP registers at abend
*                                              (registers 0-15)
PEP_COM_KEY                   DS     X         Execution key at abend
*                                              (ASRA and ASRB only)
PEP_COM_USER_KEY              EQU    9         User key
PEP_COM_CICS_KEY              EQU    8         CICS key
*
PEP_COM_STORAGE_HIT           DS     X         Storage type hit by 0C4
*                                              (ASRA only)
PEP_COM_NO_HIT                EQU    0         No hit, or not 0C4
PEP_COM_CDSA_HIT              EQU    1         CDSA hit
PEP_COM_ECDSA_HIT             EQU    2         ECDSA hit
PEP_COM_ERDSA_HIT             EQU    3         ERDSA hit
PEP_COM_RDSA_HIT              EQU    4         RDSA hit
PEP_COM_EUDSA_HIT             EQU    5         EUDSA hit
PEP_COM_UDSA_HIT              EQU    6         UDSA hit
PEP_COM_ETDSA_HIT             EQU    7         ETDSA hit
PEP_COM_GCDSA_HIT             EQU    8         GCDSA hit
PEP_COM_GUDSA_HIT             EQU    9         GUDSA hit
*
PEP_COM_SPACE                 DS     X         Subspace/basespace
PEP_COM_NOSPACE               EQU    0
PEP_COM_SUBSPACE              EQU    10        Abending task was in
*                                              subspace
PEP_COM_BASESPACE             EQU    11        Abending task was in
*                                              basespace
PEP_COM_PADDING               DS     CL2       Reserved
*
*                       Return code
*
PEP_COM_RETURN_CODE           DS     F
PEP_COM_RETURN_OK             EQU    0
PEP_COM_RETURN_DISABLE        EQU    4         Disable transaction
*
*       Additional Program status word information
*
PEP_COM_INT                   DS     CL8       PSW interrupt codes
*
*       Breaking Event Address
*
PEP_COM_BEAR                  DS     AD        Breaking Event Addr
*
*       Additional register information
*
                              DS     0D        Force alignment
```

The sample program error programs

Two source-level versions of the default program are provided: DFHPEP, coded in assembler language, and DFHPEPD, coded in C. Both are in the CICSTS52.CICS.SDFHSAMP library.

You can use an assembler-language macro, DFHPCOM, and a corresponding C copy book, DFHPCOMD, to define the communication area. These are found in the CICSTS52.CICS.SDFHMAC and CICSTS52.CICS.SDFHC370 libraries, respectively.

You can code your program error program in any of the languages supported by CICS, but you must always name it DFHPEP.

Chapter 9. Writing a transaction restart program

You can write a transaction restart user-replaceable program (DFHREST) to participate in the decision as to whether a transaction is restarted.

CICS invokes DFHREST when a transaction abends, if RESTART(YES) is specified in the transaction's resource definition (the default is RESTART(NO)).

The default program requests restart under certain conditions; for example, in the event of a program isolation deadlock (that is, when two tasks each wait for the other to release a particular DL/I database segment or file record), one of the tasks is backed out and automatically restarted, and the other is allowed to complete its update.

For general information about restarting transactions, see the Troubleshooting for recovery processing in Troubleshooting and support.

Note:

1. If your transaction restart program chooses to restart a transaction, a new task is attached that invokes the initial program of the transaction. This is true even if the task abends in the second or subsequent UOW, and DFHREST requested a restart.
2. Statistics on the total number of restarts against each transaction are kept.
3. Emergency restart does not restart any tasks.
4. In some cases, the benefits of transaction restart can be obtained instead by using the SYNCPOINT ROLLBACK command. Although use of the ROLLBACK command is not usually recommended, it does keep all the executable code in the application programs.

When planning to replace the default DFHREST, check to see if the logic of any of your transactions is inappropriate for restart.

- Transactions that execute as a single unit of work are safe. Those that execute a loop, and on each pass reading one record from a recoverable destination, updating other recoverable resources, and closing with a syncpoint, are also safe.
- There are two types of transaction that need to be modified to avoid erroneously repeating work done in the units of work that precede an abend:
  1. A transaction in which the first and subsequent units of work change different resources
  2. A transaction where the contents of the input data area are used in several units of work.
- Distributed transactions whose principal facilities are APPC links should not be considered for transaction restart. Restarting a back-end or front-end transaction while the other side of the conversation is still active presents problems with correct error handling and recovery of the conversation state.

All the following conditions must be true for CICS to invoke the transaction restart program:

- A transaction must be terminating abnormally.

- The transaction abend which caused the transaction to be terminating abnormally must have been detected before the commit point of the implicit syncpoint at the end of the transaction has been reached.
- The transaction must be defined as restartable in its transaction definition.
- The transaction must be related to a principal facility.

If these conditions are satisfied, CICS invokes the transaction restart program, which then decides whether or not to request that the transaction be restarted. CICS can subsequently override the decision (for example, if dynamic backout fails). Also, if the transaction restart program abends, the transaction is not restarted.

If these conditions are not satisfied, CICS does not invoke the transaction restart program and the transaction is not restarted.

The DFHREST communications area

The CICS-supplied default transaction restart program is written in assembler and contains logic to:

- Address the communications area passed to it by CICS
- Decide whether or not to request transaction restart
- Send a message to CSMT if restart is requested
- Return control to CICS using the EXEC CICS RETURN command.

The communications area is mapped by the `XMRS_COMMAREA` DSECT, which is supplied in the DFHXMRSD copybook. The equivalent structures for C, COBOL, and PL/I are contained in the copybooks DFHXMRSH, DFHXMRSO, and DFHXMRSP, respectively.

The information passed in the communications area is as follows:

`XMRS_FUNCTION`
Indicates, in a 1-byte field, the function code for this call to the restart program. This is always set to 1, which equates to `XMRS_TRANSACTION_RESTART`, which means that DFHREST is called to handle transaction restart.

`XMRS_COMPONENT_CODE`
Indicates, in a 2-byte field, the component code of the caller. This is always set to XM, which equates to `XMRS_TRANSACTION_MANAGER`. The transaction manager is the CICS component that coordinates the decision whether or not to restart a transaction.

`XMRS_READ`
Indicates, in a 1-byte field, whether the transaction has issued any terminal read requests, other than for initial input.

The equated values for this parameter are:

XMRS_READ_YES
Means a terminal read has been performed by the transaction.

XMRS_READ_NO
Means no terminal read has been performed.

`XMRS_WRITE`
Indicates, in a 1-byte field, whether the transaction has issued any terminal write requests.

The equated values for this parameter are:

XMRS_WRITE_YES
Means a terminal write has been performed by the transaction.

XMRS_WRITE_NO
Means a terminal write has not been performed by the transaction.

XMRS_SYNCPOINT
Indicates, in a 1-byte field, whether the transaction has performed any sync points.

The equated values for this parameter are:

XMRS_SYNCPOINT_YES
Means one or more sync points have been performed.

XMRS_SYNCPOINT_NO
Means no sync points have been performed.

XMRS_RESTART_COUNT
This indicates, as an unsigned, half-word binary value, the number of times the transaction has been restarted.

It is zero if the transaction has not been restarted. It is **not** the total number of restarts for the transaction definition. Rather it is the total number of restarts for transactions that are attempting, for example, to process a single piece of operator input.

XMRS_ORIGINAL_ABEND_CODE
Provides the first abend code recorded by the transaction.

XMRS_CURRENT_ABEND_CODE
Provides the current abend code. The values of the original abend code and the current abend code can be different if, for example, a transaction handles an abend and then abends later.

XMRS_RESTART
This is a 1-byte output field that the transaction restart program sets to indicate whether it wants CICS to restart the transaction.

The equated values for this field are:

XMRS_RESTART_YES
Requests a restart.

XMRS_RESTART_NO
Requests no restart.

The CICS-supplied transaction restart program

The CICS-supplied default transaction restart program requests that the transaction be restarted if:

1. The transaction has not performed a terminal read (other than reading the initial input data), terminal write or sync point, **and**
2. The restart count is less than 20 (to limit the number of restarts), **and**
3. The current abend code is one of the following:
   - ADCD, indicating that the transaction abended due to a DBCTL deadlock
   - AFCF, indicating that the transaction abended due to a file control-detected deadlock

- AFCW, indicating that the transaction abended due to a VSAM-detected deadlock (RLS only).

Note: Pseudoconversational transactions started with RETURN TRANSID CHANNEL() cannot be restarted.

The source of the CICS-supplied default transaction restart program, DFHREST, is supplied in assembler language only, in the CICSTS52.CICS.SDFHSAMP library.

The assembler copybook for mapping the communications area is in the CICSTS52.CICS.SDFHMAC library.

Chapter 10. Writing a terminal error program

The CICS terminal error program (TEP) handles error conditions for devices that use the sequential access method. You cannot use a terminal error program for z/OS Communications Server-supported devices. For z/OS Communications Server, use a node error program instead.

CICS provides a sample terminal error program that you can use as the basis for your own program.

Background to error handling for sequential devices

CICS terminal error handling allows you to modify CICS operations in response to terminal errors. Because CICS cannot anticipate all possible courses of action, the error-handling facilities have been designed to allow maximum freedom for users to create unique solutions for errors that occur within a terminal network.

The following CICS components are involved in the detection and correction of errors that occur when sequential devices are used:

- Terminal control program (DFHTCP)
- Terminal abnormal condition program (DFHTACP)
- Terminal error program (DFHTEP).

These components are discussed in the following sections. The corresponding CICS components for logical units are discussed in Chapter 11, "Writing a node error program," on page 465.

When an abnormal condition occurs

When an abnormal condition associated with a particular terminal or line occurs, the terminal control program puts the terminal out of service, and passes control to the terminal abnormal condition program (DFHTACP) which, in turn, passes control to a version of the terminal error program (DFHTEP, either CICS-supplied or user-written), so that it can take the appropriate action.

Terminal control program

When the terminal from which the error was detected has been put out of service, the terminal control program creates a terminal abnormal condition line entry (TACLE), which is chained off the real entry, the terminal control table line entry (TCTLE) for the line on which the error occurred. The TACLE contains information about the error.

Terminal abnormal condition program

After the TACLE has been established, a task that executes DFHTACP is attached by the terminal control program and is provided with a pointer to the real line entry (TCTLE) on which the error occurred. After performing basic error analysis and establishing the default actions to be taken, DFHTACP gives control to DFHTEP, and passes a communication area (DFHTEPCA) so that DFHTEP can examine the error and provide an alternative course of action.

The communication area provides access to all the error information necessary for correct evaluation of the error; and contains special action flags that can be manipulated to alter the default actions previously set by DFHTACP.

After DFHTEP has performed the function required, it returns control to DFHTACP by issuing an EXEC CICS RETURN command. DFHTACP then performs the actions dictated by the action flags within the communication area, and the error-handling task terminates.

Note: If DFHTACP has more than eight errors on a line before action can be taken, the line is put out of service to avoid system degradation.

Terminal error program

The terminal error program analyzes the cause of the terminal or line error that has been detected by the terminal control program. The CICS-supplied version (the sample terminal error program, DFHXTEP) is designed to attempt basic and generalized recovery actions. A user-written version of this program can be provided to handle specific application-dependent recovery actions.

The user-written terminal error program is linked-to in the same way as the CICS-supplied version, by the terminal abnormal condition program. Information relating to the error is carried in the communication area and the TACLE.

The macros that are provided for generating the sample terminal error program are described in the sections that follow. The main steps are generating the sample DFHTEP module and tables with the DFHTEPM and DFHTEPT macros, respectively. You can select the appropriate options in this sample program, and you can base your own version on it.

There is a description of the CICS-supplied sample terminal error program (DFHXTEP), and advice about how to generate a user-written version, in a later sub section.

The communication area

The communication area is the basic interface used by the sample DFHTEP, and should be used by a user-written DFHTEP to:

- Address the TACLE
- Indicate the course of action to be taken on return to DFHTACP.

Before giving control to DFHTEP, DFHTACP establishes which default actions must be taken. This depends on the particular error condition that has been detected. The default actions are indicated by appropriate bit settings in the 1- byte communication area field TEPCAACT. For details about communication area fields, default actions, and bit settings, refer to “Writing a terminal error program” on page 453.

Terminal abnormal condition line entry (TACLE)

The TACLE contains further information about the type of error, and about the type of terminal that is in error.

The code indicating the detected error condition is passed to DFHTEP in the 1-byte field of the TACLE labeled TCTLEPFL. These DFHTACP error codes, message codes, conditions, and default actions are also listed in the *CICS Problem Determination Guide*.

A format description of the terminal abnormal condition line entry (TACLE) DSECT is provided under "Writing a terminal error program" on page 453.

Sample terminal error program

CICS provides a sample terminal error program (TEP) that can be used as a generalized program structure for handling terminal errors.

Note that, although the source code form of the sample TEP (DFHXTEP) is provided in assembler language only, you can write your own terminal error program in any of the languages supported by CICS.

After DFHXTEP has been assembled, it is link-edited as DFHTEP. For information about the job control statements necessary to assemble and link-edit user-replaceable programs, refer to Chapter 6, "Assembling and link-editing user-replaceable programs," on page 419.

You can generate and use the sample terminal error program with the default options provided, or you can customize the terminal error support to the needs of the operating environment by selecting the appropriate generation options and variables. Because each error condition is processed by a separate routine, you can replace a CICS-provided routine with a user-written one when the sample TEP is generated.

Components of the sample terminal error program

The sample terminal error program consists of the terminal error program itself and two terminal error program tables:

- The TEP error table
- The TEP default table.

Both tables contain "thresholds" defined for the various error conditions to be controlled and accounted for by the sample DFHTEP. A "threshold" may be thought of as the number of error occurrences that are permitted for a given type of error on a given terminal before the sample DFHTEP accepts the DFHTACP default actions. Optionally, the number of occurrences can be controlled and accounted for over prescribed time intervals (for example, if more than three of a given type of error occur in an hour, the terminal is put out of service).

The TEP error table

The terminal error program (TEP) error table maintains information about errors that have occurred on a terminal.

The table consists of two parts (shown in Figure 30 on page 436):

- The TEP error table header (TETH), which contains addresses and constants related to the location and size of the TEP error table components.
- Terminal error blocks (TEBs), which can be either:
  - Permanent (P-TEBs), each associated with a particular terminal
  - Reusable (R-TEBs), not permanently associated with any particular terminal.

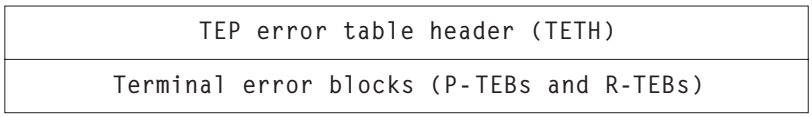


Figure 30. TEP error table

TEBs maintain error information associated with terminals. You must specify the total number of TEBs to be generated. The maximum number needed is one per terminal. In this case the TEBs are permanent.

You can reduce the total amount of storage used for TEBs by allocating a pool of reusable TEBs, that are not permanently associated with a particular terminal. Reusable TEBs are assigned dynamically on the first occurrence of an error associated with a terminal, and are released for reuse when the appropriate error processor places the terminal out of service.

Note: Ensure that the pool is large enough to hold the maximum number of terminals for which errors are expected to be outstanding at any one time. If the pool limit is exceeded, handling of terminal errors may become intermittent. **No warning is given of this condition**.

You should assign permanent TEBs to terminals that are critical to the network. For the remainder of the network, you can generate a pool of reusable TEBs.

Each TEB currently in use or permanently assigned contains the symbolic terminal identifier of the terminal, and one or more error status elements (ESEs), as shown in Figure 31.

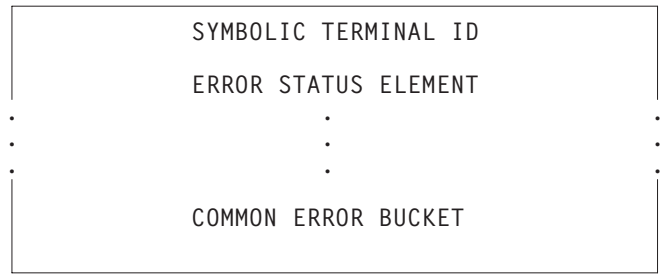


Figure 31. Terminal error block (TEB)

An ESE records the occurrence of a particular type of error associated with the terminal. The contents of an error status element are described in the TEPCD DSECT (generated by the DFHTEPM TYPE=INITIAL macro) under the comment "ERROR STATUS ELEMENT FORMAT". The number of ESEs per TEB remains constant for all TEBs. You specify the number when the TEP tables are generated. If fewer than the maximum number of error types recognized by DFHTACP (25) are specified, one additional ESE, referred to as the "common error bucket", is generated for each TEB.

You can permanently reserve ESE space in each TEB for specific error types. Those not permanently reserved are considered reusable, and are assigned dynamically on the first occurrence of a particular error type associated with the terminal. If an error type occurs that is not currently represented by an ESE, and if all reusable ESEs are assigned to other error types, the occurrence of this error is recorded in

the common error bucket. DFHTACP can recognize far more error types than can occur in a typical terminal network. By specifying less than the maximum and allowing the sample DFHTEP to assign ESEs dynamically, you can minimize the table size, and still control and account for the types of errors relevant to the network.

TEP default table

The terminal error program (TEP) default table contains the "number and time" thresholds for each type of error to be controlled and accounted for.

An index array at the beginning of the default table serves a dual function. If the value in the index is positive, then the error code has a permanently defined ESE in each TEB and the index value is the displacement to the reserved ESE. If the index value is negative, then an ESE must be assigned dynamically from a reusable ESE if one has not already been created by a prior occurrence. The complement of the negative index value is the displacement to the thresholds for the error type retained in the TEP default table.

Structure of the sample terminal error program

The structure of the sample terminal error program (DFHXTEP) can be broken down into six major areas as follows:

1. Entry and initialization
2. Terminal identification and error code lookup
3. Error processor selection
4. Error processing execution
5. General exit
6. Common subroutines.

These areas are described in detail in the sections that follow.

Figure 32 on page 440 gives an overview of the structure of the sample terminal error program.

Entry and initialization

On entry, the sample TEP uses DFHEIENT to establish base registers and addressability to EXEC Interface components.

It obtains addressability to the communication area passed by DFHTACP by means of an EXEC CICS ADDRESS COMMAREA, and addressability to the EXEC interface block with an EXEC CICS ADDRESS EIB command. It gets the address of the TACLE from the communication area, and establishes access to the TEP tables with an EXEC CICS LOAD. If time support has been generated, the error is time-stamped for subsequent processing. (Current time of day is passed in the communication area.) The first entry into the sample TEP after the system is initialized causes the TEP tables to be initialized.

Terminal ID and error code lookup

After the general entry processing, the TEP error table is scanned for a terminal error block (TEB) entry for the terminal associated with the error.

If no matching entry is found, a new TEB is created. If all TEBs are currently in use (if no reusable TEBs are available), the processing is terminated and a RETURN request is issued, giving control back to DFHTACP, where default actions are taken.

After the terminal's TEB has been located or created, a similar scan is made of the error status elements (ESEs) in the TEB to determine whether the type of error currently being processed has occurred before, or if it has permanently reserved ESE space. If an associated ESE is not found, an ESE is assigned for the error type from a reusable ESE. If a reusable ESE does not exist, the error is accounted for in the terminal's common error bucket. The addresses of the appropriate control areas (TEB and ESE) are placed in registers for use by the appropriate error processor.

Error processor selection

User-specified message options are selected and the messages are written to a specified transient data destination. The type of error code is used as an index to a table to determine the address of an error processor to handle this type of error.

If the error code is invalid, or the sample TEP was not generated to process this type of error, the address points to a routine that optionally generates an error message and returns control to DFHTACP, where default actions are taken. If an address of a valid error processor is obtained from the table, control is passed to that routine.

Error processing execution

The function of each error processor is to determine whether the default actions established by DFHTACP for a given error, or the actions established by the error processor, are to be performed.

The common error bucket is processed by the specific error processor. However, the thresholds of the common error bucket are used in determining whether the limit has been reached. Subroutines are provided in the sample TEP to maintain count and time threshold totals for each error associated with a particular terminal to assist the error processor to make its decision. Also available are subroutines for logging the status of the error and any recovery action taken by the error processor.

You can replace any of the error processors supplied in the sample TEP with user-written ones. Register linkage conventions, error conditions, DFHTACP default actions, and sample TEP error processor actions are described in comments given in the sample DFHXTEP source listing. However, sample DFHXTEP actions, in many cases, can be altered by changing the thresholds when generating the TEP tables.

General exit routine

Each error processor passes control to a general exit routine which determines whether the terminal is to remain in service.

If the terminal is to be put out of service, the terminal error block and all error status elements for that terminal are deleted from the TEP error table unless the terminal was defined as a permanent entry. When the terminal is placed back in service, a new terminal error block is assigned if a subsequent error occurs.

Common subroutines

A number of subroutines are provided in the sample DFHXTEP for use by the error processors. Each subroutine entry has a label of the form “TEP*xxxxx*” where “*xxxxx*” is the subroutine name

. All labels within a subroutine start with TEPx where “x” is the first character of the subroutine name. All subroutines are arranged within the module in

alphabetical order in the subroutine section. Register conventions and use of the subroutine are given as comments at the beginning of each subroutine in the source listing.

The following subroutines are available for writing your own error processors:

TEPACT
Used to output the names of the action bits set by DFHTACP and the sample DFHTEP in the communication area field **TEPCAACT** if appropriate PRINT options are selected when the program is generated.

TEPDEL
Used to delete the terminal error block and error status elements for a terminal from the TEP error table on exit from an error processor.

TEPHEXCN
Used by TEPPUTTD to convert a 4-bit hexadecimal value to its 8-bit printable equivalent.

TEPINCR
Used to update and test the count and time threshold totals maintained in the terminal's error status element.

TEPLOC
Used to locate or assign terminal error blocks and error status elements for a terminal ID.

TEPPUTTD
Used to output character or hexadecimal data to a user-defined transient data destination.

TEPTMCHK
Used by TEPINCR to determine whether the time threshold has been passed.

TEPWGHT
Used to update the weight/time threshold values maintained in the terminal's error status elements.

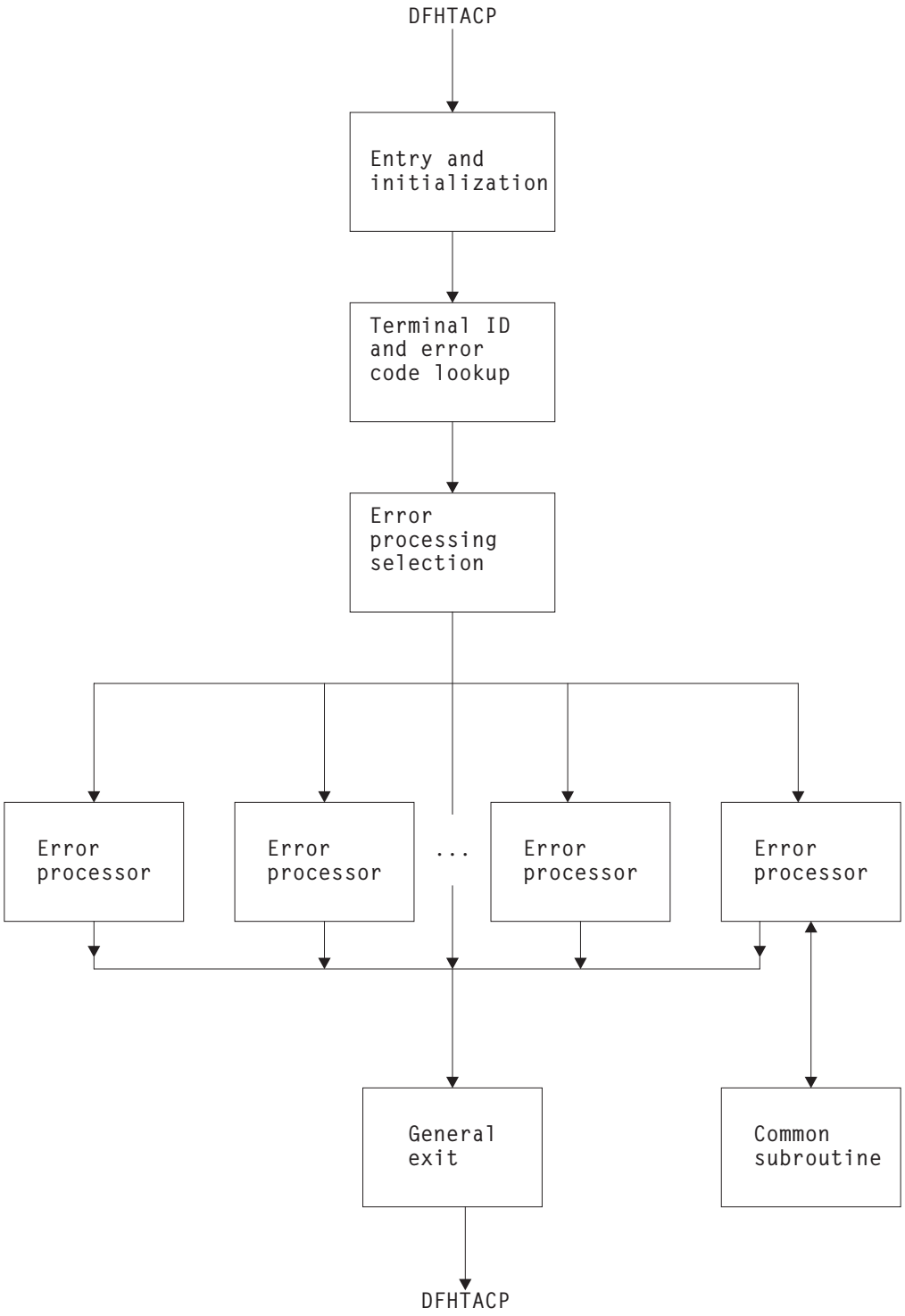


Figure 32. Overview of the sample terminal error program (DFHXTEP)

Sample terminal error program messages

The messages logged to the transient data destination CSMT (or, optionally, to the destination specified in the OPTIONS operand of DFHTEPM TYPE=INITIAL) are of six types, each identified by a unique message prefix.

You can control the selection of each type of message by using the appropriate parameters specified on the PRINT operand of DFHTEPM TYPE=INITIAL.

These messages are:

DFHTEP, ERROR – error text
During DFHTEP module generation, the PRINT parameter specified ERRORS. This message can be suppressed by using the NOERRORS option. The error text is one of the following:

Unsupported error code, "xx"
The error code presented to DFHTEP by DFHTACP is unknown to DFHTEP.

"DFHTEPT" not defined in system
The DFHTEP table could not be loaded into storage.

Unknown error status message, "xxxx"
The error status message presented from a remote 3270 type device could not be decoded.

None of these errors should occur.

DFHTEP, ACTION – action flag names
During DFHTEP module generation, the PRINT parameter specified TACPACTION or TEPACTION or both. If both are specified, this message is logged twice each time DFHTEP is called. The first message indicates the action flags as set by DFHTACP on entry to DFHTEP. The second message indicates the action flags as returned to DFHTACP by DFHTEP after error processing. These messages can be suppressed by using the NOTACPACTION and NOTEPACTION options.

The action flag names and descriptions are listed here. For further information about the actions taken by DFHTACP, see the description of the TEPCAACT field in "Addressing the contents of the communication area" on page 455.

Flag name
Description

LINEOS
Place line out of service

NONPRGT
Nonpurgeable task exists on terminal

TERMOS
Place terminal out of service

ABENDT
Abend task on terminal

ABORTWR
Abort write, free terminal storage

RELTTIOA
Release incoming message

SIGNOFF
Sign off terminal.

DFHTEP, TID - tid
During the DFHTEP module generation, the PRINT parameter specified TID.

This message contains the symbolic terminal ID of the device associated with the error. This message can be suppressed by using the NOTID option.

DFHTEP, DECB - DECB information
During the DFHTEP module generation, the PRINT parameter specified DECB. This two-line message contains the DECB (printed in hexadecimal format) of the terminal causing the error. The DECB is contained in the TACLE (displacement +16 [decimal]). See the TACLE DSECT described in "Writing a terminal error program" on page 453. This message can be suppressed by using the NODECB option.

DFHTEP, TACLE - TACLE information
During the DFHTEP module generation, the PRINT parameter specified TACLE. This message (printed in hexadecimal format) contains the first 16 bytes of the TACLE passed to DFHTEP by DFHTACP. See the TACLE DSECT described in "Writing a terminal error program" on page 453. This message can be suppressed by using the NOTACLE option.

DFHTEP, ESE - ESE information
During the DFHTEP module generation, the PRINT parameter specified ESE. This message contains the error status element. The message can be suppressed by using the NOESE option.

An ESE is either 6 bytes or 12 bytes long, depending on whether the TIME option was specified when generating the TEP tables. The formats are as follows:

Table 21. Format of error status element on DFHTEP, ESE messages—NOTIME specified

Display	Length (bytes)	Significance — NOTIME specified
0	2	Error threshold counter or weight value in binary format
2	2	Current error count or weight value in binary
4	1	Error code
5	1	Not used.

Table 22. Format of error status element on DFHTEP, ESE messages—TIME specified

Display	Length (bytes)	Significance — TIME specified
0	5	Error threshold counter or weight value in binary format
5	3	Timed threshold value in hundredths of a second
8	4	Time of first occurrence of this error. Time given as binary integer in hundredths of a second.

Generating the sample terminal error program

For information about how to generate the sample terminal error program and the sample terminal error table, refer to Chapter 6, "Assembling and link-editing user-replaceable programs," on page 419.

The sample program and tables provide you with default error processing for terminal errors. If you want to replace the supplied error processors with user-written error processors, you must use the DFHTEPM and DFHTEPT macros to generate a sample error program and tables that include your user-written routines. Some of the parameters specified in the DFHTEPM and DFHTEPT macros are related and care must be taken to ensure compatibility.

If you use the sample terminal error program (DFHXTEP), you can generate the required program and transaction definitions by using the CEDA INSTALL GROUP(DFHSTAND) command.

Job control for generating the sample terminal error program

The generation of the sample terminal error program (TEP) consists of two separate assembly and link-edit steps, one to create the sample TEP module itself, and the other to create the TEP tables.

The names under which the components must be link-edited are:

DFHTEP
Sample TEP module, assembled from DFHXTEP

DFHTEPT
Sample TEPT table, assembled from DFHXTEPT.

For information about the job control statements necessary to assemble and link-edit user-replaceable programs, refer to Chapter 6, "Assembling and link-editing user-replaceable programs," on page 419.

DFHTEPM–generating the sample DFHTEP module

The sample DFHTEP module is generated by the following macros:

- DFHTEPM TYPE=USTOR—to indicate the start of user storage definitions.
- DFHTEPM TYPE=USTOREND—to indicate the end of user storage definitions.
- DFHTEPM TYPE=INITIAL—to control the printing of CICS DSECTs, provide optional routines, and indicate the type of information to be logged when errors occur.
- DFHTEPM TYPE=ENTRY—to code a user "ENTRY" routine.
- DFHTEPM TYPE=EXIT—to code a user "EXIT" routine.
- DFHTEPM TYPE=ERRPROC—to allow you to replace the error processors supplied with the sample terminal error program with user-written versions.
- DFHTEPM TYPE=FINAL—to indicate the end of the sample DFHTEP module.

Note: You must code the translator options NOPROLOG and NOEPILOG in your error processors if you use these macros.

```
DFHTEPM  TYPE=USTOR
```

This macro indicates the start of user storage definitions. It must be followed by your storage definitions, and then by DFHTEPM TYPE=USTOREND. If you use DFHTEPM TYPE=USTOR to define storage, then both it and DFHTEPM TYPE=USTOREND must be coded **before** DFHTEPM TYPE=INITIAL.

```
DFHTEPM  TYPE=USTOREND
```

This macro indicates the end of user storage definitions. Its use is mandatory if DFHTEPM TYPE=USTOR has been coded. If you use DFHTEPM TYPE=USTOR to define storage, then both it and DFHTEPM TYPE=USTOREND must be coded **before** DFHTEPM TYPE=INITIAL.

```
DFHTEPM  TYPE=INITIAL
         [,DSECTPR={YES|NO}]
         [,OPTIONS=([TD|(TD,destid)|NOTD]
                   [,EXITS|,NOEXITS]
                   [,TIME|,NOTIME]
         [,PRINT=([ERRORS|NOERRORS]
                   [,TACPACTION|,NOTACPACTION]
                   [,TEPACTION|,NOTEPACTION]
                   [,TID|,NOTID]
                   [,DECB|,NODECB]
                   [,TACLE|,NOTACLE]
                   [,ESE|,NOESE])]
```

TYPE=INITIAL
establishes the beginning of the generation of the sample DFHTEP module itself.

DSECTPR={YES|NO}
controls the printing of CICS DSECTs on the sample DFHTEP assembly listing. Its purpose is to reduce the size of the listing. The default is DSECTPR=YES.

YES
Printing of the DSECTs is allowed.

NO Printing of selected CICS DSECTs is suppressed. This parameter should not be used under Assembler F.

OPTIONS=optional-routines
includes or excludes optional routines in the DFHTEP module. The parentheses are required even when only one option is specified. If this operand is omitted, all default options are generated.

TD|(TD, destid)|NOTD
specifies whether information regarding the errors is to be written to a transient data queue.

TD The transient data output routine is to be generated. The implied transient data queue is CSMT.

(TD, destid)
The transient data output routine is to be generated. The messages are sent to the transient data queue specified by “destid”, which must be defined to CICS with a TDQUEUE resource definition.

NOTD
No messages are to be written to a transient data queue.

EXITS|NOEXITS
specifies whether “ENTRY” and “EXIT” user routine support is to be included.

EXITS
Branches are taken to ENTRY and EXIT routines before and after error processing. Dummy routines are provided if user routines are not used.

NOEXITS
No branches are taken to user routines.

TIME|NOTIME
specifies whether threshold tests are to be controlled over prescribed time intervals. An example might be putting a terminal out of service if more than three instances of a given type of error occur in one hour. The parameter must be the same as the OPTIONS operand in the DFHTEPT TYPE=INITIAL macro.

TIME
This type of threshold testing is supported.

NOTIME
This type of threshold testing is not supported.

PRINT=print-information
specifies which types of information are to be logged to the transient data queue each time an error occurs. If NOTD is specified on the OPTIONS operand, all PRINT parameters default to NO. All PRINT parameters require the transient data output routine. The parentheses are required even when only one parameter is specified.

ERRORS|NOERRORS
specifies whether unprocessable conditions detected by the sample DFHTEP are to be recorded on the transient data queue.

ERRORS
Error messages are to be logged.

NOERRORS
No error messages are to be logged.

TACPACTION|NOTACPACTION
specifies whether DFHTACP default actions are to be recorded on the transient data queue.

TACPACTION
The default actions are logged.

NOTACPACTION
No default actions are logged.

TEPACTION|NOTEPACTION
specifies whether the actions selected as a result of sample DFHTEP processing are to be recorded on the transient data queue.

TEPACTION
The final actions are logged.

NOTEPACTION
No final actions are logged.

TID|NOTID
specifies whether the symbolic terminal ID of the terminal associated with an error is to be recorded on the transient data queue.

TID
The terminal ID is to be logged.

NOTID
No terminal IDs are to be logged.

DECB|NODECB
specifies whether the DECB of the line associated with error is to be recorded on the transient data queue.

DECB
The DECB is logged. The hexadecimal representation of the DECB is logged as two 24-byte messages.

NODECB
No DECB logging occurs.

TACLE|NOTACLE
specifies whether the TACLE prefix is to be recorded on the transient data queue.

TACLE
The 16-byte TACLE prefix as received from DFHTACP is logged.

NOTACLE
No TACLE prefix logging occurs.

ESE|NOESE
specifies whether the ESE associated with the error is to be recorded on the transient data queue.

ESE
The ESE, after being updated, and before being deleted (if the action puts the terminal out of service) is logged.

NOESE
No ESE logging occurs.

DFHTEPM TYPE=ENTRY and EXIT–for user entry and exit routines

The sample DFHTEP provides guidance about how to prepare error processor routines, particularly with regard to register and subroutine linkage conventions.

The routines must also observe the following restrictions:

- The error processor must be coded in assembler language.
- The first executable statement in the routine must be labeled TEPCDxx, where "xx" is the error code specified in the DFHTEPM TYPE=ERRPROC,CODE=errcode macro.
- Register usage conventions and restrictions are stated in the sample DFHTEP source.
- The error processor must exit to the sample DFHTEP symbolic label TEPRET.

The macro required for a user "ENTRY" routine is:

```
DFHTEPM  TYPE=ENTRY
```

This macro must be immediately followed by user "ENTRY" routine code, starting with the label "TEPENTRY" and ending with a BR 14 instruction.

The macro required for a user "EXIT" routine is:

```
DFHTEPM  TYPE=EXIT
```

This macro must be immediately followed by user "EXIT" routine code, starting with the label "TEPEXIT" and ending with a BR 14 instruction.

DFHTEPM TYPE=ERRPROC–replacing error processors

The macro necessary to replace error processors supplied with the sample DFHTEP with user-written error processors is:

```
DFHTEPM  TYPE=ERRPROC
         ,CODE=errcode
         (followed by the appropriate error
          processor source statements)
```

TYPE=ERRPROC
indicates that a CICS-supplied error processor routine is to be replaced with the user-written error processor that immediately follows the macro. This macro is optional; if used, it must follow the DFHTEPM TYPE=INITIAL macro. One DFHTEPM TYPE=ERRPROC macro must precede each user-written error processor source routine.

CODE=errcode
is used to identify the error code assigned to the appropriate error condition. These codes are listed in Figure 36 on page 459.

DFHTEPM TYPE=FINAL–ending the sample DFHTEP module

The macro to terminate the sample DFHTEP module is:

```
DFHTEPM  TYPE=FINAL
```

This is followed by an END DFHTEPNA statement.

DFHTEPM macro examples

This example generates a sample DFHTEP module with CICS-supplied error processors and all default options. This is equivalent to the CICS-supplied sample terminal error program.

1. The following is an example of the minimum number of statements required to generate a sample DFHTEP module:

   ```
   DFHTEPM    TYPE=INITIAL
   DFHTEPM    TYPE=FINAL
   END DFHTEPNA
   ```

2. Figure 33 on page 448 is an example of a more tailored sample DFHTEP module. In this example no 3270 support is generated. All default types of information except for TACP and TEP actions are to be logged to the TEPQ transient data destination. The CICS DSECTs are not printed on the sample DFHTEP assembler-language listing. There are two error processor routines (codes '87' and '9F' respectively).

```
* GENERATE USER STORAGE

   DFHTEPM     TYPE=USTOR
USORFLD    DS  F
   DFHTEPM     TYPE=USTOREND

* MODULE SPECIFICATIONS

   DFHTEPM     TYPE=INITIAL,                          *
               OPTIONS=((TD,TEPQ),NO3270,EXITS),      *
               PRINT=(NOTEPACTION,NOTACPACTION),      *
               DSECTPR=NO

* USER-SUPPLIED ERROR PROCESSORS

   DFHTEPM     TYPE=ERRPROC,CODE=87
TEPCD81    DS  0H
           -
           -   error processor "87" source statements
           -
           B   TEPRET

   DFHTEPM     TYPE=ERRPROC,CODE=9F
TEPCD9C    DS  0H
           -
           -   error processor "9F" source statements
           -
           B   TEPRET

* USER "EXIT" EXIT CODE

   DFHTEPM     TYPE=EXIT
TEPEXIT    DS  0H
           -
           -
Additional user source statements to be executed after
error processing:
           -
           -
           BR  R14

* CONCLUDE MODULE GENERATION

   DFHTEPM     TYPE=FINAL
   END DFHTEPNA
```

Figure 33. Example of DFHTEPM macros used to generate a sample DFHTEP module

DFHTEPT–generating the sample DFHTEP tables

The following macros are required to generate the terminal error program tables:

- DFHTEPT TYPE=INITIAL—to establish the control section.
- DFHTEPT TYPE=PERMTID—to define permanently reserved terminal error blocks (TEBs) for specific terminals.
- DFHTEPT TYPE=PERMCODE|ERRCODE—to define permanently reserved error status elements (ESEs).
- DFHTEPT TYPE=BUCKET—to define specific error conditions to be accounted for in the common error bucket.
- DFHTEPT TYPE=FINAL—to end the set of DFHTEPT macros.

DFHTEPT TYPE=INITIAL–establishing the control section

The DFHTEPT TYPE=INITIAL macro necessary to establish the control section for the TEP tables is:

```
DFHTEPT  TYPE=INITIAL
         ,MAXTIDS=number
         [,MAXERRS={25|number}]
         [,OPTIONS={TIME|NOTIME}]
```

TYPE=INITIAL
establishes the beginning of the generation of the TEP tables.

MAXTIDS=number
specifies the total number of permanent and reusable terminal error blocks to be generated in the TEP error table. Permanent entries are defined by the DFHTEPT TYPE=PERMTID macro described later in this section. Any entries not defined as permanent are reused when the terminal is taken out of service, or are deleted at the request of an error processor. If an error occurs, and no TEB space is available, the error is not processed, and DFHTACP default actions are taken. The minimum number of blocks is 1. A maximum number is not checked for but should be no greater than the number of terminals in your network.

MAXERRS=25|number
specifies the number of errors to be recorded for each terminal. This value determines the number of permanent and reusable error status elements in each TEB. The maximum number that can be specified is 25 (the default value). If more are requested, only the maximum are generated. If fewer are requested, one extra ESE is generated for each TEB. The extra ESE is the common error bucket. Permanently reserved ESEs are defined by the DFHTEPT TYPE=PERMCODE macro described later in this section. Any ESEs not defined as permanent are dynamically assigned on the first occurrence of a nonpermanent error type associated with the terminal. By defining a number less than the maximum, and allowing the sample DFHTEP to assign ESEs dynamically, you can minimize the size of the table and still control and account for the error types relevant to the network. The minimum number that can be specified is zero. In this case only a common error bucket is generated.

OPTIONS={TIME|NOTIME}
specifies whether time threshold space is to be reserved in support of the TIME option specified in the DFHTEPM TYPE=INITIAL macro. The default is OPTIONS=TIME.

TIME
Time threshold space is reserved.

NOTIME
Time threshold space is not reserved.

DFHTEPT TYPE=PERMTID–assigning permanent terminal error blocks

Use the DFHTEPT TYPE=PERMTID macro to define permanently reserved terminal error blocks for specific terminals.

Syntax

```
DFHTEPT  TYPE=PERMTID
         ,TRMIDNT=name
```

TYPE=PERMTID
defines permanently reserved terminal error blocks for specific terminals.

Permanent TEBs are defined for terminals that are critical to system operation to ensure that error processors are always executed in the event of errors associated with that terminal. If no permanent TEBs are to be defined, this macro is not required. A separate macro must be issued for each permanently reserved TEB. The maximum number of permanent TEBs is the number specified in the MAXTIDS operand of the DFHTEPT TYPE=INITIAL macro.

TRMIDNT=name
is used to provide the symbolic terminal ID (1 - 4 characters) for a permanently defined TEB. Only one terminal can be specified in each macro.

DFHTEPT TYPE=PERMCODE|ERRCODE–defining error status elements

The DFHTEPT TYPE=PERMCODE | ERRCODE macro is used to change the default threshold constants of the sample DFHTEP, and to define permanently reserved error status elements:

```
DFHTEPT  TYPE={PERMCODE|ERRCODE}
         ,CODE={errcode|BUCKET}
         [,COUNT=number]
         [,TIME=(number[,SEC|,MIN|,HRS])]
```

TYPE={PERMCODE|ERRCODE}
identifies whether the error code specified in the macro is to have a permanently reserved or a dynamically assigned ESE. These macros are required only if permanently reserved ESEs are to be defined, or if the sample DFHTEP default threshold constants are to be overridden. These are listed in Table 23 on page 452.

PERMCODE
Identifies the error code specified as having a permanently reserved ESE. Each permanently reserved ESE must be identified by a separate DFHTEPT TYPE=PERMCODE macro. All DFHTEPT TYPE=PERMCODE macros must precede all DFHTEPT TYPE=ERRCODE macros.

ERRCODE
Indicates that the error code specified does not require a permanently reserved ESE, but that the sample DFHTEP default threshold constants are to be changed. Each error code requiring a threshold constant change, other than those defined as permanently reserved, must be identified by a separate DFHTEPT TYPE=ERRCODE macro. All DFHTEPT TYPE=ERRCODE macros must follow all DFHTEPT TYPE=PERMCODE macros.

CODE={errcode|BUCKET}
identifies the error code referred to by the TYPE=PERMCODE | ERRCODE parameter. These codes are listed in Figure 36 on page 459. CODE=BUCKET is only applicable to the DFHTEPT TYPE=ERRCODE macro. It is used to override the default threshold constants established for the common error bucket.

COUNT=number
can be used in either the DFHTEPT TYPE=PERMCODE or TYPE=ERRCODE macro to override the sample DFHTEP default count threshold (see Table 23 on page 452). When the number of occurrences of the error type specified reaches the threshold, an error processor normally takes a logic path that causes DFHTACP default actions to be taken. If the number of occurrences is less than the threshold, the error processor normally takes a logic path that overrides the DFHTACP default actions. The updating and testing of the current threshold

counts are normally performed by a DFHTEP subroutine that sets a condition code that the error processor can test to determine whether the limit has been reached. **If you specify 0 as the number in the COUNT operand, you are not told when the threshold is reached**.

TIME=(number[,SEC|,MIN|,HRS])
can be used in either the DFHTEPT TYPE=PERMCODE or TYPE=ERRCODE macros to override the sample DFHTEP default time threshold (see Table 23 on page 452). This operand is only applicable when OPTIONS=TIME is specified on both the DFHTEPM and DFHTEPT TYPE=INITIAL macros. When the number of occurrences reaches the threshold specified on the COUNT operand (as already mentioned) within the interval specified on this parameter, an error processor normally takes a logic path that causes DFHTACP default actions to be taken. If the number of occurrences within the interval is less than the threshold, the error processor normally takes a logic path that overrides the DFHTACP default actions. If the time interval has expired, the sample DFHTEP subroutine that normally updates and tests the current threshold count resets the occurrence counts, and establishes a new expiration time. In this case, the condition code set by the subroutine indicates that the thresholds had not been reached.

Time control in the sample DFHTEP starts with the first occurrence of an error type. Subsequent occurrences of the same error type **do not** establish new starting times, but are accounted for as having occurred within the interval started by the first occurrence. This continues until an error count reaches the threshold within the interval started by the first occurrence, or until the interval has expired. In the latter case, the error being processed becomes a first occurrence, and a new interval is started. A time interval of 0 means that the number of occurrences is to be accounted for and controlled without regard to a time interval. Zero is the implied time interval if the value of the COUNT operand is 0 or 1. It is also the implied time interval if the time options are not generated.

The time interval can be expressed in any one of four units; hours, minutes, seconds, or hundredths of a second. The maximum interval must be the equivalent of less than 24 hours. A practical minimum would be 1 to 2 minutes. This allows for access method retries and the time required to create the task to service each error. The four methods of expressing the threshold time interval are:

number
The interval in units of one hundredth of a second. Parentheses are not required if this method is used. The maximum number must be less than 8,640,000 (24 hours).

(number,SEC)
The interval in whole seconds, which must be enclosed in parentheses. The maximum number must be less than 86,400 (24 hours).

(number,MIN)
The interval in whole minutes, which must be enclosed in parentheses. The maximum number must be less than 1,440 (24 hours).

(number,HRS)
The interval in whole hours, which must be enclosed in parentheses. The maximum number must be less than 24.

Table 23 illustrates the default thresholds of the sample terminal error program, referred to in the TYPE, COUNT, and TIME operands of the DFHTEPT TYPE=PERMCODE | ERRCODE macro.

Table 23. Default thresholds of the sample TEP

CODE=	COUNT=	TIME=
81	3	(7,MIN)
84	1	0
85	1	0
87	50 (Note 2)	0
88	1	0
8C	1	0
8D	1	0
8E	1	0
8F	1	0
90	0	0
91	0	0
94	7	(10,MIN)
95 (Note 1)	0	0
96	2	(1,MIN)
97 (Note 1)	0	0
99	1	0
9F (Note 1)	0	0
BUCKET	5	(5,MIN)

Notes:

1. The error processor maintains an error count only. DFHTACP default actions are always taken regardless of the thresholds.
2. The error processor uses a threshold "weight" instead of a threshold count (see the source code of the sample DFHTEP).

DFHTEPT TYPE=BUCKET–using the error bucket for specific errors

The DFHTEPT TYPE=BUCKET macro is used to ensure that specific error conditions are always accounted for in the common error bucket:

```
DFHTEPT  TYPE=BUCKET
         ,CODE=errcode
```

TYPE=BUCKET
generates the macro to account for specific error conditions in the common error bucket. If MAXERR=25 on the DFHTEPT TYPE=INITIAL macro, this macro cannot be used. This macro is not required if no error codes are to be specifically accounted for in the common error bucket. Each error code must be identified by a separate DFHTEPT TYPE=BUCKET macro.

CODE=errcode
identifies the error code to be specifically accounted for in the common error bucket. The error code must not be specified in the DFHTEPT TYPE=PERMCODE or TYPE=ERRCODE macro.

DFHTEPT TYPE=FINAL–terminating DFHTEPT entries

The DFHTEPT TYPE=FINAL macro terminates the generation of the DFHTEP tables.

Syntax

```
DFHTEPT  TYPE=FINAL
```

DFHTEPT–examples of how the macros are used

This example generates 10 reusable terminal error blocks, each capable of accounting for the maximum number of error types. Time threshold control is supported, and all threshold values are the defaults supported by the sample DFHTEP. This is equivalent to the CICS-supplied sample terminal error program.

1. The following is an example of the minimum number of statements required to generate the TEP tables:

   ```
   DFHTEPT    TYPE=INITIAL,MAXTIDS=10
   DFHTEPT    TYPE=FINAL
   END
   ```

2. Figure 34 is an example of a customized TEP table (continuation characters omitted).

```
* TABLE SPECIFICATIONS

       DFHTEPT     TYPE=INITIAL,MAXTIDS=10,
                   MAXERRS=5

* PERMANENT TERMINAL DEFINITIONS

       DFHTEPT     TYPE=PERMTID,TRMIDNT=TM02

* PERMANENT ERROR CODE DEFINITIONS

       DFHTEPT     TYPE=PERMCODE,CODE=81
       DFHTEPT     TYPE=PERMCODE,CODE=87,
                   COUNT=2,TIME=(1,MIN)

* OTHER THRESHOLD OVERRIDES

       DFHTEPT     TYPE=ERRCODE,CODE=BUCKET,
                   COUNT=3,TIME=(3,MIN)

* CONCLUDE TABLE GENERATION

       DFHTEPT     TYPE=FINAL
       END
```

Figure 34. Example of the use of DFHTEPT macros to generate DFHTEP tables

This example generates 10 terminal error blocks, one of which is reserved for the terminal whose symbolic ID is TM02, and the other nine are reusable. Each TEB has space for five error status elements plus a common error bucket. Of the five ESEs, two are reserved for error codes '81' and '87'; the remaining ESEs are available to be assigned dynamically. The thresholds for error code '87' and the common error bucket are being changed. No specific error code is to be accounted for in the common error bucket.

Writing a terminal error program

You can write your own terminal error program (TEP) in any of the languages supported by CICS.

CICS-supplied code is provided in assembler language only. The names of the supplied source files and macros, and the libraries in which they can be found, are listed in Table 24.

Table 24. Supplied source files and macros

Name	Type	Description	Library
DFHXTEP	Source	Sample terminal error program (assembler language)	CICSTS52.CICS.SDFHSAMP
DFHXTEPT	CSECT	Sample terminal error tables (assembler language)	CICSTS52.CICS.SDFHSAMP
DFHTEPM	Macro	Sample TEP program generator (assembler language)	CICSTS52.CICS.SDFHMAC
DFHTEPT	Macro	TEP table generator (assembler language)	CICSTS52.CICS.SDFHMAC
DFHTEPCA	Macro	assembler language communication area	CICSTS52.CICS.SDFHMAC

The user-written DFHTEP receives control in the same manner as the CICS-supplied sample DFHTEP, described in "Background to error handling for sequential devices" on page 433. It should therefore use the communication area as its basic interface with DFHTACP.

Why write your own terminal error program?

- There are some situations in which CICS may try to send a message to an input-only terminal; for example, an 'invalid transaction ID' message, or a message wrongly sent by an application program. You should provide a terminal error program to reroute these messages to a system destination such as CSMT or CSTL or other destinations, by means of transient data or interval control facilities.
- There could be application-related activity to be carried out when a terminal error occurs. For example, if a message is not delivered to a terminal because of an error condition, it may be necessary to notify applications that the message needs to be redirected.
- Not all errors represent communication-system failures; for example, SAM end-of-data conditions.

Restrictions on the use of EXEC CICS commands

The commands that a terminal error program (TEP) can issue are restricted. In particular, you should avoid commands that require a principal facility, as they cause unpredictable results.

Do not use commands that invoke the following functions:

- Terminal control ("CEMT-type" commands, such as **EXEC CICS INQUIRE TERMINAL**, *are* permissible)
- BMS (except routing)
- ISC communication (including function shipping).

Addressing the contents of the communication area

After your terminal error program receives control from DFHTACP, it should obtain the address of the communication area by means of an EXEC CICS ADDRESS COMMAREA command.

About this task

You generate the communication area DSECT by coding DFHTEPCA TYPE=DSECT in your program. The layout of the communication area is shown in Figure 35.

```
                        IN/OUT
                         PARM

                      0XL4              Standard Header
TEPCALDS     DS       XL1        I      Function Code        Always '1'
TEPCAGDS     DS       XL2        I      Component Code       Always 'TC'
             DS       XL1               Reserved
TEPCATCA     DS       A          I      Address of TACLE being processed
TEPCECIA     DS       A          I      Address of TCTUA
TEPCECIL     DS       H          I      Length of TCTUA
TEPCAACT     DS       XL1       I/O     User action byte
TEPCATID     DS       CL4        I      Terminal identity
TEPCATDB     DS       F          I      Current time of day binary
```

Figure 35. The DFHTACP/DFHTEP communication area

The parameter list contains the following information:

TEPCALDS
Function Code. The Function Code is a printable character representing the identity of the task within the TCP which invoked DFHTEP. It always has the value '1'.

TEPCAGDS
Component Code. This always has the value 'TC', representing a component of the TCP.

TEPCATCA
Contains the address of the TACLE being processed.

TEPCECIA
Contains the address of the terminal control table user area (TCTUA).

TEPCECIL
Contains the length of the TCTUA.

TEPCAACT
The User action byte. One of the main uses of the communication area is to transmit the actions that are to be taken for a terminal. TEPCAACT contains the following flags, which can be reset within DFHTEP:

LINEOS (X'80')
Place line out of service

NONPRGT (X'40')
Nonpurgeable task exists on the terminal

TERMOS (X'20')
Place terminal out of service

ABENDT (X'10')
Abend the task on the terminal

ABORTWR (X'08')
Abend write, free terminal storage

RELTTIOA (X'04')
Release TCAM incoming message. (TCAM is no longer supported.)

SIGNOFF (X'02')
Call sign-off program.

On entry to DFHTEP, these flags represent the default actions set by DFHTACP. The write-abend bit (communication area field **ABORTWR**) and the abend-task bit (communication area field **ABENDT**) are always set if the place-line-out-of-service bit (X'80') is set; but both bits are suppressed if "dummy terminal" is indicated (see "Resetting the flags in the user action byte, TEPCAACT").

On return to DFHTACP, the flags represent the actions as modified by DFHTEP.

TEPCATID
Contains the identity of the terminal in error.

TEPCATDB
Contains the time of day when the error occurred, in binary format.

Resetting the flags in the user action byte, TEPCAACT

About this task

The following factors should be considered when altering the action bits in TEPCAACT:

- You should consider how to preserve data security. For example, if a terminal is put out of service for some time (until the cause of the failure is removed) the signon information is still in the TCTTE when the terminal is put back into service, although the original operator may no longer be present. To prevent a possible security violation, you can set the **SIGNOFF** bit to sign off the terminal.
- The dummy terminal indicator at TCTLEPF2 is set on errors from which no specific terminal is indicated. Therefore, if a dummy terminal is indicated, abend task and abend write are not set. The dummy terminal is only used to identify the line.
- The abend-task bit (X'10' in TEPCAACT) is always associated with two other bits as part of TACP's abend transaction processing. These other bits are nonpurgeable task and abend write (X'40' and X'08' respectively, both in TEPCAACT).
- Abend write is always set on at the same time as abend task. It has the effect of clearing the TCTTE of the original write request indicators, if the error being processed occurred on a TC WRITE.
- Nonpurgeable task is set on if a transaction is currently associated with the terminal, and the transaction ID was specified with TPURGE=NO.

None of the abend-task, abend-write, or nonpurgeable-task bits is set if the dummy terminal indicator is on, even if DFHTACP would normally set abend task as the default for the error being processed. Therefore, the following remarks apply only to errors related to a real terminal.

- Abend task has no effect if no transaction is associated with the terminal; (except where a pseudoconversational task is associated with the terminal, in which case, the next transid is cleared). Otherwise, if nonpurgeable task is indicated, the transaction remains attached to the terminal (normally in SUSPEND state) and DFHTACP writes the 'DFHTC2522 INTERCEPT REQUIRED' message to CSMT; if the task is not marked nonpurgeable, it is abended with code 'AEXY' or, rarely, 'AEXZ'.
- Abend write has no effect if the TCTTE was associated with a READ request. In this case the normal result is that, if the line and terminal remain in service, the read is retried.

Addressing the contents of the TACLE

The TACLE is created by the terminal control program when the error occurs, and contains all the I/O error information provided by BSAM.

About this task

To address the contents of the TACLE, the user-written terminal error program should contain the COPY DFHTACLE and COPY DFHTCTLE statements, in that order. These define the complete DFHTCTLE DSECT. The symbolic names in this DSECT are used to address fields in both the TACLE and the real line entry associated with the error.

The TACLE consists of a 16-byte prefix (defined by COPY DFHTACLE) and a further 48-byte section, which is a modified copy of the DECB of the real line entry at the time the TACLE was created.

To address the TACLE, the user-written terminal error program should therefore contain the statements:

```
COPY DFHTACLE
COPY DFHTCTLE

L TCTLEAR,TEPCATCA        POINT TO TACLE
USING DFHTCTLE,TCTLEAR
```

Note that fields normally part of the real line entry DECB have offsets increased by 16 in the TACLE.

The following fields in the DECB copy in the TACLE do **not** represent data copies from the real line entry:

```
TCTLEDCB                (Offset 24 in TACLE,
                         8 in real TCTLE)
```

This field in the TACLE points to the real line entry; in the real line entry, it points to the BSAM DCB for the line group.

```
TCTLECSW                (Offsets 46, 48 in TACLE,
TCTLEALP                 30, 32 in real TCTLE)
```

These are used in the TACLE for SAM error information.

The following statements give direct addressability to the **real** line entry:

```
COPY DFHTCTLE
COPY DFHTCTTE

L      TCTLEAR,TEPCATCA        POINT TO TACLE
USING  DFHTCTLE,TCTLEAR
```

```
L      TCTTEAR,TCTLEPTE        POINT TO ERROR TCTTE
USING  DFHTCTTE,TCTTEAR
DROP   TCTLEAR
L      TCTLEAR,TCTTELEA        POINT TO TCTLE
USING  DFHTCTLE,TCTLEAR
```

After you have carried out the required functions and, optionally, altered the default actions scheduled by DFHTACP, the user-written DFHTEP must return control to DFHTACP by issuing the EXEC CICS RETURN command. DFHTACP then performs the actions specified in the TACLE and causes the error processing task to terminate.

The format of the TACLE DSECT is shown in Figure 36 on page 459.

TERMINAL ABNORMAL CONDITION LINE ENTRY

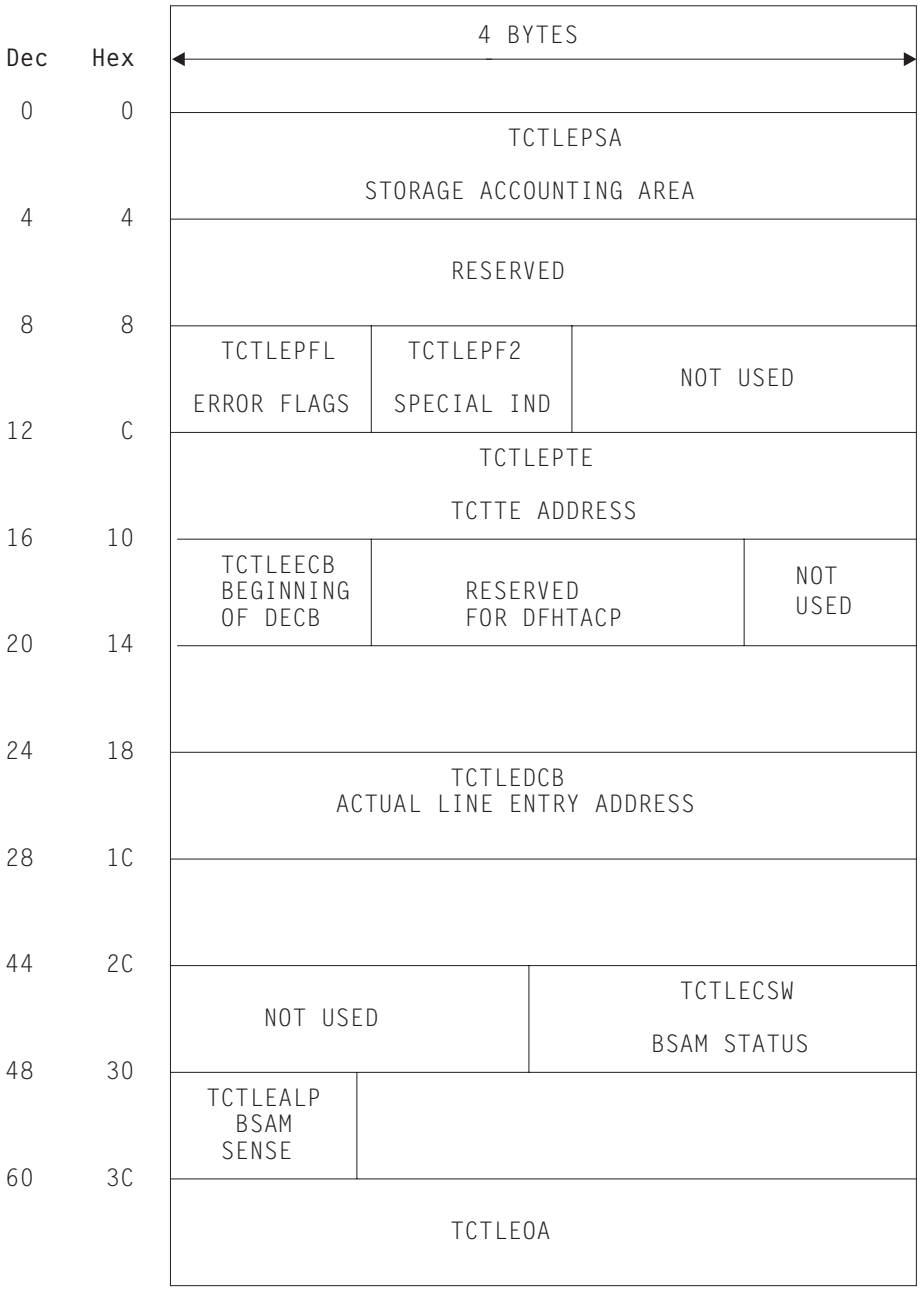


Figure 36. Format description of the TACLE DSECT (part 1)

```
Displacement

Dec     Hex   Code     Bytes Label       Meaning


0       0             4     TCTLEPSA    Storage accounting

                            RESERVED

8       8             1     TCTLEPFL    Error flags
              81                          Message too long

              84                          TCT search error
              85                          Write not valid
              87                          Unsolicited input
              88                          Input event rejected
              8C                          Output event rejected
              8D                          Output length of zero
              8E                          No output area
              8F                          Output area exceeded
              94                          Unit check
              95                          Unit check
                                          (should not occur)
              96                          Unit exception
              97                          Unit exception
                                          (should not occur)
              99                          Undetermined I/O error
              9F                          Invalid destination
                                          (TCAM: no longer supported)
              .
              .      (All codes not listed are reserved and are
              .       not intended for use by DFHTEP)
              .

9       9             1     TCTLEPF2    Special indicator
              01                          dummy terminal

12      C             4     TCTLEPTE    Address of terminal
                                          entry for terminal
                                          in error

16      10            4     TCTLEECB    DECB/copy of line
                                          when error occurred

60      3C            4     TCTLEOA     For TCAM lines only.
                                          (No longer supported)
```

Figure 37. Format description of the TACLE DSECT (part 2)

Example of a user-written terminal error program

The "DFHTEP recursive retry routine" on page 461 is an example of the logic steps necessary to design a portion of the terminal error program. In Figure 38 on page 462, 10 retries are provided for each terminal; however, the logic could be used for any number of retries. The following assumptions are made:

USER FIELD A
(PCISAVE)
represents a 6-byte field in the process control information (PCI) area of the TCTTE. This field is used to preserve the count of input and output from the TCTTE when the first error occurs. These counts are contained in 3-byte fields located at TCTTENI and TCTTENO within the TCTTE.

USER FIELD B

(PCICNT)
represents a user-defined field used to accumulate the count of recursive errors. It should be in the process control information (PCI) area of the TCTTE.

SYSTEM COUNT
(TCTTENI)
represents the 6-byte field in the TCTTE that contains the terminal input and output counts (TCTTENI+TCTTENO). In the example, these two adjacent fields are considered as one 6-byte field.

Because this example requires access to the TCT terminal entry (TCTTE) to examine the SYSTEM COUNT and to locate the process control information (PCI) area, the DFHTCTTE symbolic storage definition is included so that fields can be symbolically referenced.

DFHTEP recursive retry routine

```
*ASM     XOPTS(NOPROLOG NOEPILOG SP)
*********************************************************************
*                                                                   *
*                   DFHTEP RECURSIVE RETRY ROUTINE                  *
*                                                                   *
*********************************************************************
         DFHEISTG
         DFHEIEND
         DFHTEPCA TYPE=DSECT       COMMAREA passed by TACP
         COPY  DFHA06DS            Statistics DSECT
         USING DFHA06DS,STATBAR
PCIAREA  DSECT
PCISAVE  DS    XL6                 User Field A
PCICNT   DS    PL2                 User Field B
*
TCTLEAR  EQU   2                   Pointer to TACLE
STATBAR  EQU   4                   Pointer to statistics DSECT
TCTUABAR EQU   5                   Pointer to TCTUA
COMMABAR EQU   12                  Pointer to COMMAREA passed by TACP
         EJECT
DFHTEP   CSECT
*********************************************************************
*        Establish addressability                                   *
*********************************************************************
         DFHEIENT
*
         EXEC CICS ADDRESS EIB(11)
*
         EXEC CICS ADDRESS COMMAREA(COMMABAR)
*
         USING DFHTEPCA,COMMABAR
         L     TCTLEAR,TEPCATCA    Load TACLE address
*
         USING PCIAREA,TCTUABAR
         L     TCTUABAR,TEPCECIA   Load TCTUA address
*
*********************************************************************
*        Start processing                                           *
*********************************************************************
         TM    PCICNT+1,X'0C'      Has User Field B been initialized
*                                   to a packed decimal number?
         BO    CKCOUNT             YES .... so compare the system count
*                                   with the existing count in Field B
RESET    DS    0H
         MVC   PCICNT,=PL2'+0'     NO  .... so initialize field B to
*                                   packed zero.
*
```

Figure 38. DFHTEP recursive retry routine (part 1)

```
         EXEC CICS COLLECT STATISTICS TERMINAL(TEPCATID) SET(STATBAR)
*                                Get statistics for this terminal
*                                using TERMID passed in Commarea
*
         MVC   PCISAVE,A06TENI    Save the current system counts. This
*                                is a new error, or first time
*                                through.
INCR     DS    0H
         AP    PCICNT,=P'1'       Increment the number of times this
*                                error has occurred (recursive count)
*
         CP    PCICNT,=P'10'      Has the maximum recursive error
*                                limit been reached?
         BNE   RETRY              NO .... set action
*
         ZAP   PCICNT,=P'0'       Clear and reset user fields for next
*                                error set
         EXEC CICS COLLECT STATISTICS TERMINAL(TEPCATID) SET(STATBAR)
*                                Get statistics for this terminal
*                                using TERMID passed in COMMAREA
*
         MVC   PCISAVE,A06TENI    Get current system counts
         B     NORETRY            Action indicators for no retry
*
CKCOUNT  DS    0H
         EXEC CICS COLLECT STATISTICS TERMINAL(TEPCATID) SET(STATBAR)
*                                Get statistics for this terminal
*                                using TERMID passed in COMMAREA
*
         CLC   PCISAVE,A06TENI    Has system count changed since last
*                                entry to TEP?
         BNE   RESET              YES .... this is a new error since
*                                some I/O activity has occurred on
*                                terminal.
         B     INCR               NO  .... this is a recursive error,
*                                so increment the recursive count and
*                                check for retry.
RETRY    DS    0H
*                                The user would include here the code
*                                necessary to alter the flags in the
*                                COMMAREA so that a retry can be
*                                performed on the terminal.
NORETRY  DS    0H
*                                The user would include here the code
*                                necessary to allow DFHTACP to take
*                                final actions on the terminal; that
*                                is, abend task, put line out of
*                                service, and others.
         LTORG ,
         END
```

Figure 39. DFHTEP recursive retry routine (part 2)

Note that the code in Figure 38 on page 462 is intended only as an illustration of a recursive error handling technique and of the steps necessary to establish addressability to the applicable control blocks.

Chapter 11. Writing a node error program

You can write a node error program (NEP) for terminals and logical units that are supported using the ACF/SNA interface.

CICS supplies a sample node error program that you can use as the basis for your own program. Like the terminal error program for non-z/OS Communications Server devices, the node error program for SNA-attached terminals is available in three forms:

1. The default node error program
2. The CICS-supplied sample node error program
3. User-written versions.

If you code an **EXEC CICS HANDLE CONDITION TERMERR** command in your application program, it is sometimes possible for the application program to handle exceptional cases, rather than using a node error program. The TERMERR condition is driven if the node abnormal condition program (DFHZNAC) actions an ABTASK (ATNI abend). The TERMERR condition is application-related and is not an alternative to the node error program, which must be used for session-related problems. Dealing with errors in the application program is particularly useful in an intersystem communication (ISC) environment.

Background to CICS-z/OS Communications Server error handling

Errors detected by CICS-z/OS Communications Server LU control are queued for handling by a special task, the CICS node error handler (transaction CSNE).

CICS uses the same task for some housekeeping work, such as sending “good morning” messages, and logging session starts and ends, which are not errors.

In a few cases, exceptions signaled to CICS by z/OS Communications Server are not treated as errors, and are not passed to the node error handler. For example, CICS often sends a z/OS Communications Server BID command as part of automatic transaction initiation. Rejection of the BID with exception code ‘0813' (wait) is a standard response, and CICS handles the retry in terminal control without calling this an error. In the rest of this description, only the errors are considered.

The CSNE task runs as a “background” task, meaning that it is not associated with any one CICS terminal. At any time, there is at most one such task, working on the single node error queue.

All node errors on the queue are analyzed in turn by a table-driven, CICS-supplied program called DFHZNAC (node abnormal condition program). It is not intended that you should ever modify this.

DFHZNAC links to a module called DFHZNEP (if present in the CICS system) when processing most node errors. (It does not link to DFHZNEP for errors that are not related to a specific node—for example, those caused by a z/OS Communications Server shutdown.) The interface for this link is described in “When an abnormal condition occurs” on page 472. This formal DFHZNAC to DFHZNEP interface gives you the opportunity to supply your own code to

analyze error conditions, change default actions by setting various “action flags”, and take additional actions specific to your applications.

CICS supplies a pregenerated default DFHZNEP, which sets the “print TCTTE” action flag if a z/OS Communications Server storage problem is detected, and returns control to DFHZNAC. Because it leaves all other action flags unchanged, DFHZNAC's default actions are not otherwise affected. (DFHZNAC's default actions for different error conditions are listed in Appendix C, “Default actions of the node abnormal condition program,” on page 895.)

Why use a NEP to supplement CICS default actions?

For a variety of reasons, you might want to write your own node error program.

The following list gives some of the reasons why you might want to write your own node error program to add to the default actions provided by CICS and the z/OS Communications Server.

- Not all errors represent communication system failures. Some errors (such as trying to write zero-length data) may reflect special situations in applications, needing special action.
- You might want to output extra data, in addition to the error messages sent by DFHZNAC. (Note that you cannot use the node error program to suppress messages from DFHZNAC.) All data output from DFHZNAC and DFHZNEP is written to the transient data queue CSNE.
- In other cases, you might want to change the amount of diagnostic information produced by CICS: the default varies with the error type. For example, the z/OS Communications Server RPL associated with an error may be printed when you do not want it, or not printed when you do.
- There could be application-related activity to be performed when a node error occurs. For example, if a message fails to be delivered to a terminal, it may need redirecting to another. With messages sent with exception-response only, CICS may not have the data available to send it again, but the requesting application might be able to re-create it. For example, if an error were signaled during the sending of a document to a printer, it might be able to restart from the beginning, or from a specific page.
- Some devices, such as the 3650 Retail Store System, return application-type data in “User Sense Data” fields. This can only be retrieved in a NEP. The NEP has to catch and save data for further application programs.

An overview of writing a NEP

Your DFHZNEP module must conform to the defined interface: that is, it must be a linked-to program that uses defined communication area fields to analyze an error and then returns to DFHZNAC. The source code of the default NEP provided by CICS can be used as a skeleton on which to build a single NEP.

CICS also provides macros to help you generate more complex sample NEPs. These are **aids** to help you develop your own NEPs; you do not have to use any of them.

Your error-handling logic can be written as a number of modules, but the one that receives control from DFHZNAC must be called DFHZNEP.

DFHZNEP code can use standard CICS functions (LINK, XCTL) to invoke other user modules. Each module thus requested must have either an installed CSD

program definition or an autoinstalled program definition. Program resource definitions for DFHZNAC and DFHZNEP themselves are provided in the IBM-supplied CSD group, DFHVTAM.

The key features of the DFHZNAC - DFHZNEP interface are as follows:

- DFHZNEP can be written in any of the CICS-supported languages.

  Note: CICS-supplied NEP code is provided in assembler language only. The communication area parameter list is supplied in assembler-language and C versions.
- DFHZNEP is linked-to separately for each node-related error on the queue. (Note that, because sense codes are always associated with an error, DFHZNEP is not linked-to separately for these.)
- Communication between the two modules is through a communication area (DFHNEPCA).

The structure of the communication area is described in "The communication area" on page 473.

On each DFHZNEP invocation, one field in the communication area contains a 1-byte internal error code, assigned by DFHZNAC, which identifies the type of error. Other fields identify the CICS TCTTE (LU) associated with the error, and any SNA sense codes. There are also fields for DFHZNEP to pass back user messages for subsequent logging by DFHZNAC.

Further fields contain "action flags". Each flag represents an action that DFHZNAC may take when DFHZNEP returns control to it. These actions are of different types:

- Reporting (dumps of control blocks, actions taken)
- Status changes (for example, of TCTTE)
- Clean-up work (cancel any associated transaction, end the z/OS Communications Server session).

The action flags can be set or reset within DFHZNEP.

The action flags set by DFHZNAC for specific error codes and sense codes are listed in Appendix C, "Default actions of the node abnormal condition program," on page 895.

The default NEP

The CICS-supplied default NEP, DFHZNEP, sets the "print TCTTE" action flag (TWAOTCTE in the user option byte TWAOPT1; see "The user option bytes (TWAOPTL)" on page 476) if a z/OS Communications Server storage problem is detected; otherwise it performs no processing, leaves the action flags set by DFHZNAC unchanged, and returns control to DFHZNAC.

The sample NEP

The sample node error program (NEP) is a generalized program structure for handling errors detected from logical units.

None of the sample NEP's components is generated as part of the standard CICS generation process, but instead may be optionally generated as described in this section and in "Sample node error program" on page 480.

The sample NEP that CICS provides is designed with two main features:

- It assumes that you want to invoke separate user-supplied error processors to handle different "groups" of error types. You specify which of the DFHZNAC internal error codes are to be regarded as a "group" for processing by any one routine, and then supply the code for that routine. CICS has some standard cases to help you.
- The supplied error processors may work in association with a separately generated module called a node error table. This can be used to build up statistics for each error group that the NEP processes. This table is analogous to the terminal error table, DFHTEPT, used by the sample terminal error program.

  Some of the CICS-supplied error processors use the node error table—for example, that for errors affecting 3270 LUs (GROUP=1) (see "DFHSNEP TYPE=DEF3270—including error processors for 3270 LUs" on page 485).

The node error table

To understand the sample NEP, first look at the node error table structure in more detail.

Node error table is often abbreviated to NET. You should not confuse this acronym with "net" (as in "network"), or with a NETNAME.

You can generate a node error table using the CICS macro DFHSNET. See "Node error table" on page 482 and "DFHSNET—generating the node error table" on page 488. You choose how complex this table is to be.

The node error table must be defined as a RESIDENT program. This makes it easy for the NEP to find it (using a CICS LOAD request), and ensures that any counters are not reset by reloading. You can give the table any name you like. The default is DFHNET.

The table consists of sets of error-recording areas. Each set is called a node error block (NEB) and is used to count node errors relating to a single LU. You can dedicate specific NEBs to specific LUs throughout a CICS run; and you can leave other, reusable NEBs for general use. If you expect to accumulate error statistics about 10 LUs concurrently, you need 10–12 NEBs.

Each NEB may contain multiple recording areas, one being used for each group of errors you want to distinguish. The error groups correspond to those in the NEP. That is, they are groups of error types requiring separate processing logic.

Each recording area is known as an error status block (ESB). You specify the space reserved for each ESB, and it typically includes space to count the errors, or record when the first of the present series occurred. Note that in any one NEB the counting is for one LU only.

Finally, you can specify a threshold count and a time limit in the table. These are constants that can be used by code in the NEP to test an ESB, to see if a given type of error has occurred more than the threshold number of times in the stated interval. The time limit also affects the interval between using a general NEB for one LU and then reusing it for another.

A minimal NET would consist of a handful of NEBs, each with just one ESB, grouping together all types of error that are of interest.

Coding the sample NEP

The sample NEP is coded using the macro DFHSNEP.

The basic form is as follows:

```
DFHSNEP TYPE=INITIAL

Specific error handling code. For example:

DFHSNEP TYPE=DEF3270

DFHSNEP TYPE=FINAL
END     DFHNEPNA
```

By default, this generates a module called DFHZNEP, which works with a node error table called DFHNET. If you want to use another table, you could code NETNAME=MYTABLE after TYPE=INITIAL. Details of the DFHSNEP macro are given in "Generating the sample node error program" on page 484.

To understand the sample code, generate a standard NEP, as with TYPE=DEF3270, shown in "DFHSNEP TYPE=DEF3270—including error processors for 3270 LUs" on page 485, and look at the resulting assembler-language listing. Here is a description of the code.

The INITIAL and FINAL macros generate the basic skeleton of the NEP. This comprises some initialization code and some common routines. All the code is built round the assumption that you have a node error table as previously described.

The initial code first tests the internal error code passed from DFHZNAC to see if it belongs to a group that the NEP needs to handle. (The groups are identified by the code you supply between the DFHSNEP INITIAL and FINAL macros. This is described in "Generating the sample node error program" on page 484.) If the particular error code is not of interest to the NEP, control is returned at once to DFHZNAC, to take default actions.

Otherwise, the relevant node error table is located by a CICS LOAD request. (As previously explained, this table should be resident in virtual storage.) The NEP code will then locate the correct ESB within a selected NEB. The latter may be permanently dedicated to the LU in error (a named NEB), or may be one taken from the general pool.

The initial code then invokes the appropriate user logic for that error group. The initial code also sets up pointers to the communication area, the NEB, and the ESB. For details, see "Generating the sample node error program" on page 484.

The common routines in the NEP provide common services for your own logic. They count and time stamp errors in the ESB, and test whether error thresholds have been exceeded. They are not documented outside the sample listings. You can generate a NEP without them if you prefer.

Your own code is inserted between the DFHSNEP TYPE=INITIAL and TYPE=FINAL macros.

Note: If the user code you insert between the DFHSNEP macros contains EXEC CICS commands, you must translate the commands, and enter the *translated code* between the DFHSNEP macros.

Each section of user logic, intended to handle a particular group of error types, is headed by a macro of the type:

```
DFHSNEP TYPE=ERRPROC,CODE=(ab,cd,...),GROUP=n
```

where X'ab', X'cd',... are the DFHZNAC internal error codes you want to process, and n is the number of the error group, and therefore also of the corresponding ESB, within a NEB, in the node error table. Successive DFHSNEP TYPE=ERRPROC macros should use groups 1, 2, 3, and so on.

The DFHSNEP TYPE=ERRPROC macros serve several purposes. They:

- Inform the NEP generation how many error groups there are
- Show which error types are to be included in each group
- Introduce the code for each group.

Note that any one DFHZNAC error code should only figure in one error group, and that any code not mentioned is ignored by the NEP. You follow each DFHSNEP TYPE=ERRPROC macro with your own logic. This should begin with standard code to save registers, or set up addressability, which is best copied from sample NEP listings.

CICS provides some standard error processors to handle specific errors on two different types of LU. These are for non-SNA 3270s (BSC 3270s attached to CICS-z/OS Communications Server), and for interactive SNA logical units like a 3767. More information is given in “When an abnormal condition occurs” on page 472.

The code for non-SNA 3270s can be generated by coding

```
DFHSNEP TYPE=DEF3270
```

where you would otherwise code a DFHSNEP TYPE=ERRPROC macro plus logic of your own. In effect, TYPE=DEF3270 defines two error groups, and associates each with an error processor. The first group comprises the four DFHZNAC error codes X'D9', X'DC', X'DD', and X'F2'. The second group contains only error code X'42', corresponding to the ‘unavailable printer' condition, a specific exception condition signaled when CICS cannot allocate a printer in response to a 3270 print request.

The 3270 sample code is not intended to cover all error conditions. Note that the code is not suitable for SNA 3270s (LU session type 2). Error conditions arising from these result in different DFHZNAC error codes and may require different handling.

You may find that the CICS-supplied code is not sufficient for other, application-related, reasons. Perhaps you want to try to reacquire lost sessions after a time interval. The code supplied for the 3767 covers only one error group with one DFHZNAC error code, X'DC', which may occur under contention protocol.

You can use these CICS-supplied error processors to generate a valid DFHZNEP listing, for tutorial purposes, without having to write any user code.

You should be aware of the following limitations of this NEP design:

- Any error types you have not allowed for are ignored by the NEP, and not accumulated into error buckets.

- You may want to handle a particular situation whenever it arises, even though DFHZNAC may assign it different error codes in different situations. For example, on an SNA 3270, switching in and out of TEST state generates status X'082B' (presentation-space integrity lost). This might result in one of several DFHZNAC error codes.

In the sample NEP structure, you would need either to test for this last case in separate error processors, or group all the DFHZNAC error codes together. If you wrote your own NEP code from scratch, you would , on entry to your NEP, test the communication area field containing the status.

Multiple NEPs

You can define a NEP transaction class that applies to every transaction that uses a particular profile, session, or terminal-type.

To do this you use the NEPCLASS attribute of a PROFILE, SESSIONS, or TYPETERM resource. (Note that any value of NEPCLASS that you specify in a PROFILE resource overrides any specified in a SESSIONS or TYPETERM resource.) NEPCLASS is a 1-byte binary field containing a value in the range 0–255. The purpose of NEPCLASS is that, while a transaction is running on the LU, you can obtain a special version of node error handling, suitable for that transaction. (This is sometimes called a “transaction-class error routine”.) The default value NEPCLASS(0) indicates that no NEPCLASS is in effect.

The DFHZNEP that gets control from DFHZNAC must test the NEPCLASS in effect at that time for the LU associated with the error. Then it either transfers control to a suitable module (the actual NEP), or branches to a specific bit of code within itself.

The DFHZNEPI macros (see “DFHZNEPI macros” on page 493) generate a DFHZNEP module that is purely a routing module. This inspects the NEPCLASS in effect for the node error passed by DFHZNAC, and transfers control (links) to another module, the real NEP, according to a NEPCLASS/name routing table built up by the macros.

If no NEPCLASS is in effect (equivalent to CEDA DEFINE PROFILE NEPCLASS(0)), or the NEPCLASS is not in the routing table, a default module is invoked. You must specify the name of this in the DFHZNEPI TYPE=INITIAL macro. (See “DFHZNEPI TYPE=INITIAL—specifying the default routine” on page 493.) If you do not specify the name, no module is invoked.

You also have to provide the sub-NEPs for the various NEP transaction classes, including, of course, one for the default NEPCLASS(0). Each of these sub-NEPs needs a separate program definition. You have the same choice in coding each sub-NEP as you had when there was just one; you can code your own, or use the CICS sample macro DFHSNEP. If you use DFHSNEP, note that there is another operand on the DFHSNEP TYPE=INITIAL macro, NAME=, which means that the generated module can be given any name you choose (to match the DFHZNEPI routing). You can use a different node error table with each sub-NEP.

Before you start using NEP routing, consider the following:

- The association of an LU (TCTTE) with a transaction NEPCLASS is only valid for about the time that the CICS task exists. Errors detected after a CICS task has ended (for example, because of a problem with a delayed output message) may not be associated with the NEPCLASS of the creating transaction.

Another problem can occur when CICS is about to start a new task for the LU as a result of an internal request from another CICS task (by an EXEC CICS START request, for example). This is usually called automatic transaction initiation. Before the task is started, CICS has to open a fresh session if none exists, by issuing a z/OS Communications Server SIMLOGON request, and then, as mentioned earlier, send a BID command. The intended task is not attached until all this is completed successfully. The NEPCLASS is not picked up from the transaction definition until then. This means that any errors arising in the ATI process (perhaps an error on BIND or BID) occur before the NEPCLASS is correctly set, so they may get routed to the default NEP and not the one for the NEPCLASS. This complicates the total node error handling for the application.

As an example, consider an application that contacts unattended programmable controllers overnight in order to read in the day's input. Recovery design in such an application is fundamental, and has to allow for errors both in ATI and in file transmission. To separate these into two NEPs could be an unnecessary complication.

- The extra development effort for a NEP split on a NEPCLASS basis might not be justified. Generally, if logic is to be split, it is on an LU basis (programmable controllers may be running applications other than 3270).

To conclude this overview, remember that the CICS sample NEPs are a good source of ideas for you to write your own NEPs, but they might not be the ideal framework for your particular needs. It is recommended that you write straightforward NEPs at first.

When an abnormal condition occurs

The following CICS components are involved when an abnormal condition is detected from a logical unit:

- The terminal control program z/OS Communications Server for SNA section: DFHZCA, DFHZCB, DFHZCC, DFHZCP, DFHZCQ, DFHZCW, DFHZCX, DFHZCY, and DFHZCZ.
- The node abnormal condition program, DFHZNAC.
- The CICS-supplied default node error program, DFHZNEP, or your own version of it.

For logical units, all information concerning the processing state of the terminal is contained in the TCTTE and the request parameter list (RPL). Consequently, when a terminal error must be handled for a logical unit, the TCTTE itself is placed onto the system error queue.

DFHZNAC assumes that system sense codes are available upon receipt of an exception response from the logical unit. Thus, analysis is performed to determine the reason for the response. Decisions, such as which action flags to set and which requests are needed, are made based upon the system sense codes received. If sense information is not available, default action flags are set, and DFHZEMW is scheduled to send a negative response, if a response is outstanding, with an error message to the terminal.

The action flags set by DFHZNAC on receipt of specific inbound system sense codes are listed in Appendix C, “Default actions of the node abnormal condition program,” on page 895.

Before executing the specified routines, DFHZNAC links to DFHZNEP. You can use DFHZNEP to perform additional error processing beyond that performed by

DFHZNAC; or to alter the default actions previously set by DFHZNAC. You need to code a node error program only if you want to do either of these things.

The action flags, set by DFHZNAC to assist the node error program, are in field TWAOPTL of the communication area.

If you want to modify DFHZNAC's actions following an abnormal situation, DFHZNEP can interrogate field TWAOPTL and modify the bit settings. If you agree with DFHZNAC's proposed actions, field TWAOPTL remains unaltered.

In most cases, DFHZNEP can modify DFHZNAC's proposed actions. The only time that DFHZNAC overrides DFHZNEP's modification of field TWAOPTL is when a logical unit is to be disconnected from CICS; that is, when DFHZNAC determines that the abnormal situation requires that CICS issue the ACF/SNA CLSDST macro for a logical unit. In such a case, DFHZNAC disconnects the terminal and abnormally terminates the task, even if DFHZNEP tries to block such actions.

Resetting of the task termination flag by the node error program is also ignored if a negative response has been sent to a logical unit, or if DFHZEMW is to write an error message to the logical unit.

When the node error program has performed its functions, it returns control to DFHZNAC by an **EXEC CICS RETURN** command.

When control is returned from DFHZNEP, DFHZNAC performs the actions specified in field TWAOPTL (except when disconnecting logical units, as noted), issuing messages and setting error codes, as necessary.

The communication area

After DFHZNEP receives control from DFHZNAC, it obtains the address of the communication area by means of an **ADDRESS COMMAREA** API command.

Figure 40 illustrates the general structure of the communication area.
The significance of each section of the communication area is described here:

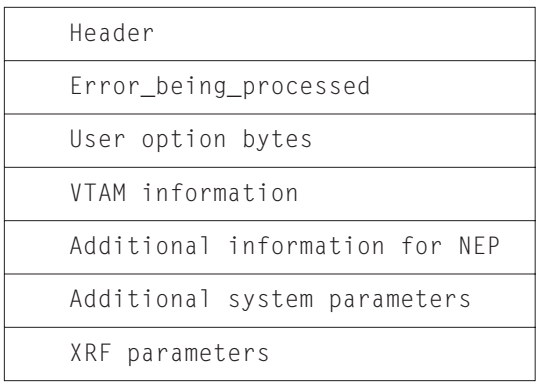


Figure 40. General structure of the communication area

Header
A 4-byte header common to all user-replaceable programs.

Error_being_processed
Identifiers of the error code and the terminal associated with the error.

User option bytes
Flags that indicate the default actions set by DFHZNAC, and that may be reset within DFHZNEP.

z/OS Communications Server information
Sense and RPL codes.

Additional info. for NEP
Other useful information for the NEP.

Additional system parameters
Locations of indirect parameters, such as the TCTTE, and other system information.

XRF parameters
Recovery notification data. The fields in TWAXRNOT can be reset by the NEP

A detailed listing of the communication area is given in Figure 41.

```
*******************************************************************************
**                       Header                                              **
**               These fields are READ ONLY                                  **
*******************************************************************************
NEPCAHDR DS    0XL4            Standard Header
NEPCAFNC DS    XL1             Function Code      Always '1'
NEPCACMP DS    XL2             Component Code     Always 'ZC'
         DS    XL1             Reserved
*******************************************************************************
**                    Error_being_processed                                  **
**    Identity of terminal and the error code associated with it             **
**               These fields are READ ONLY                                  **
*******************************************************************************
TWAEC    DS    XL1             Error Code
         DS    CL3             Reserved
TWANID   DS    CL4             Terminal identity
TWANETN  DS    CL8             Netname
*******************************************************************************
**                    User option bytes                                      **
**                 Initially set to the default actions.                     **
**                 DFHZNEP can change the defaults.                          **
*******************************************************************************
TWAOPTL  DS    0XL3            User option bytes
TWAOPT1  DS    XL1             User option byte 1
TWAOPT2  DS    XL1             User option byte 2
TWAOPT3  DS    XL1             User option byte 3
         DS    XL1             Reserved
```

Figure 41. The DFHZNAC/DFHZNEP communication area (part 1)

```
*************************************************************************
**     z/OS Communications Sever information - Any sense and RPL codes  **
**                   These fields are READ ONLY                        **
*************************************************************************
TWAVTAM  DS    0XL12           z/OS Communications Sever information
TWARPLCD DS    H               z/OS Communications Sever RPL feedback codes
         DS    H               Reserved
TWASENSS DS    0F              Sense codes to be sent
TWASS1   DS    XL1             System sense byte No 1
TWASS2   DS    XL1             System sense byte No 2
TWAUS1   DS    XL1             User sense byte No 1
TWAUS2   DS    XL1             User sense byte No 2
*
TWASENSR DS    0F              Sense codes received
TWASR1   DS    X               System sense byte No 1
TWASR2   DS    X               System sense byte No 2
TWAUR1   DS    X               User sense byte No 1
TWAUR2   DS    X               User sense byte No 2
*
*************************************************************************
**             Additional information for the NEP                      **
**Except for TWANPFW, TWANLD, and TWANLDL these fields are READ ONLY    **
*************************************************************************
TWAADINF DS    0XL22
         DS    F               Reserved
TWACTLB  DS    X               General use control byte
*        EQU   X'80'           Reserved
*        EQU   X'40'           Reserved
TWACSC   EQU   X'20'           Clear sense code indicator
TWAPSC   EQU   X'10'           Print z/OS Communications Sever sense codes
TWATIOA  EQU   X'08'           Print portion of I/O area
*        EQU   X'04'           Reserved
TWAVTRTC EQU   X'02'           z/OS Communications Sever return code available
TWANEPR  DS    XL1             NEP return code byte
TWANPFW  EQU   X'80'           Retry write with FORCE=YES
TWAREASN DS    XL1             z/OS Communications Sever reason code
TWASTAT  DS    XL1             z/OS Communications Sever status code
TWATRSN  DS    XL1             CICS terminal control
*                              terminal error code
TWAXRSN  DS                    Exception response seq number recd
TWAR     EQU   *
TWAPFLG  DS    XL1             CLSDST pass flag
TWAPIP   EQU   X'80'           CLSDST pass in progress
TWANEPC  DS    XL1             NEP class flag
TWAEISAB DS    XL1             Stand-alone begin bracket indicator
TWAESAB  EQU   X'04'           Stand-alone begin bracket
         DS    XL3             Reserved
TWANLD   DS    A               Address of data to be logged
TWANLDL  DS    H               Length of data to be logged
```

Figure 42. The DFHZNAC/DFHZNEP communication area (part 2)

```
*************************************************************************
**           Additional system parameters                              **
** Except for TWAPNETN, TWAPNTID, TWAUPRRC these fields are READ ONLY   **
*************************************************************************
TWASYSPM DS    0XL68
TWATCTA  DS    AL4             Address of TCTTE being processed
TWARPL   DS    AL4             Address of z/OS Communications Sever RPL
TWATIOAA DS    AL4             Address of data portion of TIOA
TWATIOAL DS    H               Length of data portion of TIOA
TWACOMML DS    H               Length of commarea data for TCTTE
TWACOMMA DS    CL4             Address of commarea data for TCTTE
TWATECIA DS    AL4             Address of TCTTE user area
TWATECIL DS    H               Length of TCTTE user area
TWAPPNTN DS    CL8             Primary 3270 printer netname
TWAPPTID DS    CL4             Primary 3270 printer termid
TWAPPELG DS    X               Primary printer eligible indicator
TWAPPELY EQU   X'01'            Primary printer is eligible flag
TWASPNTN DS    CL8             Secondary 3270 printer netname
TWASPTID DS    CL4             Secondary 3270 printer termid
TWASPELG DS    X               Secondary printer eligible indicator
TWASPELY EQU   X'01'            Secondary printer is eligible flag
TWAPNETN DS    CL8             Selected 3270 printer netname
TWAPNTID DS    CL4             Selected 3270 printer termid
TWAUPRRC DS    B               Unavailable Printer return code
TWAUPRNP EQU   X'00'            No printer selected
TWAUPRPS EQU   X'01'            Printer selected
TWAUPRDD EQU   X'FF'            Data disposal complete
TWAUPRPE EQU   X'FE'            Error on Put request
TWAERRF1 DS    B               Error flag byte 1
TWALXS   EQU   X'80'            Logon crossed simlogon
         DS    XL2             Reserved
*************************************************************************
**                    XRF parameters                                   **
**           XRF recovery notification data                            **
**           DFHZNEP can change these default actions                  **
*************************************************************************
TWAXRNOT DS    X               Recovery notification options
TWAXRNON EQU   X'80'            Recov notification = none
TWAXRMSG EQU   X'40'            Recov notification = message
TWAXRTRN EQU   X'20'            Recov notification = transact.
         DS    XL3             Reserved
TWAXMSTN DS    CL8             Recovery mapset name
TWAXMAPN DS    CL8             Recovery map name
TWAXTRAN DS    CL4             Recovery transaction ID
*
```

Figure 43. The DFHZNAC/DFHZNEP communication area (part 3)

The next sections describe fields in the parameter list that can be reset within DFHZNEP. See also "Coding for the 3270 'unavailable printer' condition" on page 491, which describes the use of the flags in the "unavailable printer return code" field.

The user option bytes (TWAOPTL)

TWAOPTL contains the user option bytes TWAOPT1, TWAOPT2, and TWAOPT3, each of which contains action flags. On entry to DFHZNEP, these flags represent the default actions previously set by DFHZNAC. They can be reset by DFHZNEP.

TWAOPT1

User option byte 1. TWAOPT1 contains flags which are principally debugging aids. The first five flags cause DFHZNAC to write the desired information to the CSNE log if the appropriate bit is set. Setting the sixth flag (TWAODNTA) on causes CICS to take a system dump when there is no task attached to the terminal at the time of error detection, if the flag TWAOAT in TWAOPT2 is also set on. Setting the TWAONQN flag causes the network qualified name to

be printed after any message that contains the action flag. Similarly setting the TWAOTNA flag causes the TNADDR information to be printed.

The flags are:

TWAOAF (X'80')
Print action flags.

TWAORPL (X'40')
Print z/OS Communications Server RPL.

TWAOTCTE (X'20')
Print TCTTE.

TWAOTIOA (X'10')
Print TIOA.

TWAOBIND (X'08')
Print BIND area.

TWAODNTA (X'04')
System dump if no task attached.

TWAONQN (X'02')
Print NQNAME.

TWAOTNA (X'01')
Print TNADDR (TCP/IP client address, port and, optionally, host name).

Note:

1. Note that DFHZC2411 is not related to a specific node—that is, the TCTTE has not yet been created, and the message is printed against a dummy TCTTE. The node error program is not called in this case, therefore the default setting cannot be overridden. This means that the NQNAME and the TNADDR information is always printed for DFHZC2411 messges.
2. When DFHZC2410 is issued against the dummy TCTTE, the NQNAME and TNADDR are not printed.

TWAOPT2
User option byte 2. TWAOPT2 contains flags which are task-related.

The NEP can abend the task by setting TWAOAT, or cancel it by setting TWAOCT. The difference is that abend task does not take effect until the task requests or completes a terminal control operation: cancel task takes effect as soon as system and data integrity can be maintained. Setting TWAOAT to abend the task is normally sufficient, except where the task performs lengthy processing (such as a database browse) between terminal requests. If both TWAOAT and TWAOCT are set, TWAOCT (cancel task) takes priority.

If the task is to be abnormally terminated, sends and receives are purged. If TWAOGMM is set, the next transid is cleared and any communication area associated with the terminal is released—except in the case of permanent transids (specified on the TERMINAL definition as TRANSACTION(name)), when the communication area is not released. If the TYPETERM of the terminal indicates that the “good morning” message is supported (LOGONMSG(YES)), if TWAONINT is off, and if the terminal is not in a BMS paging session, then the “good morning” message transaction is initiated (the transaction specified by the system initialization parameter GMTRAN).

The flags are:

TWAOAS (X'80')
Abandon any SEND for this terminal

TWAOAR (X'40')
Abandon any RECEIVE for this terminal

TWAOAT (X'20')
Abend any task attached to TCTTE

TWAOCT (X'10')
Cancel any task attached to TCTTE

TWAOGMM (X'08')
"good morning" message to be sent

TWAOPBP (X'04')
Purge any BMS pages for this session

TWAOASM (X'02')
SIMLOGON required.

Note:

1. If a definite response SEND has been performed, CICS has to issue a RECEIVE in order to obtain the response. If the response is negative, DFHZNAC is entered and sets flags TWAOAS (abandon the SEND) and TWAOAR (abandon the RECEIVE). TWAOAR must be kept on to ensure that the RECEIVE for the response is abandoned.
2. If the request is to be retried, and the break connection action flag is off (that is, if TWAOCN in TWAOPT3 is off), then one or more of TWAOAS, TWAOAR, and TWAONEGR must be off as well as TWAOAT.
3. The abend code returned as a result of setting TWAOCT is unpredictable.
4. TWAOGMM forces TWAOAT only if set on by the node error program.
5. TWAOPBP forces TWAOAT to be set on.
6. For non-pipeline terminals, TWAOAT acts as a cancel request (TWAOCT) if the task has not yet been dispatched for the first time.

TWAOPT3
User option byte 3. TWAOPT3 contains flags which are node-related.

The flags are:

TWAOINT (X'80')
Internally generated logons (INTLOGs) allowed

TWAONINT (X'40')
No internally-generated logons allowed. Do not set this flag when processing error codeX'49' (TCZCLSIN)

TWAONCN (X'10')
Normal CLSDST (no reset allowed)

TWAOSCN (X'08')
Normal CLSDST (reset allowed)

TWAONEGR (X'04')
Send negative response

TWAOOS (X'02')
Keep node out of service

---

1. Do not set this flag when processing error code X'49' (TCZCLSIN).

TWAOCN (X'01')
CLSDST node. Do not set this flag when processing error codeX'49' (TCZCLSIN).

TWAONINT forces TWAOCN.

TWAONEGR forces TWAOAR and TWAOAT.

TWAOOS forces TWAOCN.

TWAOCN forces TWAOAR, TWAOAS, and TWAOAT.

TWAOOS indicates that no further processing is to be done for this node. The node is logically out of service.

For an LU6.1 intersystem communication session, TWAOOS or TWAONINT causes the system entry to be put out of service if, as a result of the specified action, there are no allocatable sessions remaining. (A session can also be put out of service because of either an unknown modename being passed to z/OS Communications Server during an attempt to bind an APPC session, or an invalid logmode name for a z/OS Communications Server 3270-type terminal. However, the CICS default action resulting from this condition cannot be overridden in the NEP.)

If TWAOCN is set, the task is abnormally terminated and communication with the node is lost. Note that the NEP cannot reset this flag.

TWAOSCN provides the same function as TWAONCN, but the NEP can reset it if the session is not to be closed.

If DFHZNAC is scheduled because of the receipt of an exception response, the sense information in the TCTTE is available to DFHZNAC and DFHZNEP to determine any necessary actions.

If DFHZNAC is scheduled because of loss of the connection between CICS and a logical unit, DFHZNAC abnormally terminates any transaction in progress at the time of the failure. DFHZNEP and transaction-class error routine analysis and processing are permitted, but you should not attempt to retry the message.

However, if the application program handles the 'TERMERR' condition, the transaction is not abended. Control is returned to the program. In this circumstance, no further use can be made of the failed session.

Additional information for the NEP (TWAADINF)

Fields TWANPFW, TWANLD, and TWANLDL can be reset by the NEP.

For information about the use of TWANPFW, see the supplied sample node error program, and "Optional error processor for interactive logical units" on page 484.

TWANLD and TWANLDL — using the DFHZNAC logging facility:

You can use the logging facility available in DFHZNAC to help you retrieve information from fields TWANLD and TWANLDL.

You specify the address of the data that you want to examine in field TWANLD of the communication area, and the length of the data in field TWANLDL. The data is logged to the CSNE transient data queue for future inspection.

Note: No data in excess of 220 bytes is logged.

You can also send user-written messages to the CSNE log using the transient data facility. To write your messages, you must code the EXEC CICS WRITEQ TD instruction directly into the node error program.

TWAPIP — and application routing failure:

You can use the EXEC CICS ISSUE PASS command to pass control from CICS to another named z/OS Communications Server application. By using the ISSUE PASS command you can invoke the z/OS Communications Server macro CLSDST with OPTCD=PASS to notify CICS of the outcome of your CLSDST requests.

For programming information about the EXEC CICS ISSUE PASS command, see ISSUE PASS, in the *CICS Application Programming Reference* manual. The ISSUE PASS command in turn invokes the z/OS Communications Server macro CLSDST with OPTCD=PASS, and, in addition, if NOTIFY has been specified on the CLSDSTP system initialization parameter, with PARMS=(THRDPTY=NOTIFY). CICS is then notified of the outcome of any CLSDST request.

This notification results in an informative message being issued, and causes DFHZNAC to invoke your NEP, whether the CLSDST request has failed or succeeded. The NEP can discover that a CLSDST OPTCD=PASS request is in progress by examining field TWAPFLG for the pass-in-progress indicator, TWAPIP. The success or failure of the CLSDST OPTCD=PASS request can be determined by examining the error code at TWAEC.

If the pass operation fails, DFHZNAC sets up a default set of recovery actions that can be modified by your NEP. A possible recovery, when, for example, the target application program is not active, would be to reestablish the session with the initial application using a SIMLOGON request and for CICS to send its “good morning” message to the terminal. The default action is to leave the session disconnected and to make it NOCREATE.

If CLSDSTP=NONOTIFY has been specified, and autoinstall is being used, CICS takes no action, even if the ISSUE PASS fails.

If persistent sessions support is active, autoinstall terminals are deleted after the AIRDELAY, so any expected NEP processing as a result of CLSDSTP=NOTIFY being coded does not take place.

The additional system parameters (TWASYSPM)

If a data element referenced in this section of the parameter list (for example, the TIOA) does not exist when the NEP is driven, its address and length fields are set to zero.

Fields TWAPNETN, TWAPNTID, and TWAUPRRC can be reset by the NEP.

Sample node error program

The sample node error program provides a general environment for the execution of error processing routines (error processors), each of which is specific to certain error codes generated by the node abnormal condition program.

Sufficient optional error processors for normal operation of interactive logical unit networks are provided; these can be easily supplemented or replaced by user-supplied error processors.

There are three types of error that may occur in an SNA network:

- Errors in the host system
- Communication errors, such as session failures
- Abnormal conditions at the terminal, such as intervention required and invalid requests.

A sample node error program is supplied with CICS, and can be used as the basis of each subsequent node error program that you write. This provides you with:

- A general environment within which your error processing programs can be added
- The default node error program in a system that has several node error programs.

The CICS-supplied sample node error program is described in greater detail in the following topics.

Compatibility with the sample terminal error program

Receipt of sense or status codes corresponds to error codes X'D9', X'DC', X'DD', and X'F2'. Weighted counts of these messages are maintained against numeric and time thresholds. If the numeric threshold is exceeded, default actions are taken. If the time threshold is reached, the count is reset. This is equivalent to the function in the sample TEP, except that sense or status arising out of the "from" device on a COPY command is now presented to the node error program as an error on the "to" device; this causes the threshold to be exceeded, resulting in the request being terminated, although the terminal remains in service. Some of the weights for errors that occur on the 3270 display device have been revised, but otherwise the weight and threshold values are the same as the defaults used in the sample TEP. Time threshold maintenance for the sample NEP is mandatory, and not optional as in the sample TEP.

For further information about time and threshold count limits, see the information about the sample terminal error program in Chapter 10, "Writing a terminal error program," on page 433.

The 3270 message 'unavailable printer' corresponds to error code X'42' (interval control PUT request has failed). The algorithm used for printer selection differs in z/OS Communications Server support. The retry algorithm in the sample node error program is similar to this new selection algorithm.

Components of the sample node error program

The sample node error program comprises the following components:

- An entry section.
- The routing mechanism.
- The node error table.
- Optional common subroutines.
- Optional error processors for 3270 or interactive logical units. A node error program cannot be generated with both 3270 and interactive logical unit error processors.

The components are described in the sections that follow.

Entry section

On entry, the sample NEP uses DFHEIENT to establish base registers and addressability to the EXEC interface. It uses an **EXEC CICS LOAD PROGRAM** command to establish addressability to the node error table (NET) and, if included, the common subroutine vector table (CSVT).

It uses an **EXEC CICS ADDRESS COMMAREA** command to obtain addressability to the communication area passed by DFHZNAC, and an **EXEC CICS ADDRESS EIB** command to obtain addressability to the EXEC interface block. If time support has been generated, the error is time-stamped for subsequent processing.

Routing mechanism

The routing mechanism invokes the appropriate error processor depending on the error code provided by the node abnormal condition program.

Groups of one or more error codes are defined in the DFHSNEP macro. Each group is associated with an index (in the range X'01' through X'FF') and an error processor. A translate table is generated and the group index is placed at the appropriate offset for each error code. Error codes not defined in groups have a zero value in the table. An error processor vector table (EPVT) contains the addresses of the error group processors, positioned according to their indexes. The vector table extends up to the maximum index defined; undefined intermediate values are represented by zero addresses.

The error code is translated to obtain the error group index. A zero value causes the node error program to take no further action; otherwise the index is used to obtain the address of the appropriate error processor from the EPVT. A zero address causes the node error program to take no further action; otherwise a call is made to the error processor. This is entered with direct addressability to the NET and CSVT areas. When the error processor has been executed, the node error program returns control to the node abnormal condition program.

Node error table

The node error program may use a node error table (NET) that comprises the node error blocks (NEBs) used to maintain error status information for individual nodes (see Figure 44 on page 483). Some or all of the NEBs can be permanently reserved for specific nodes; others are dynamically assigned to nodes when errors occur. Dynamically assigned NEBs are used exclusively for the nodes to which they are assigned until they are explicitly released. All the NEBs have an identical structure of error status blocks (ESBs). Each ESB is reserved for one error processor and associated with it by means of the appropriate error group index. The ESB length and format can be customized to the particular error processor that it serves.

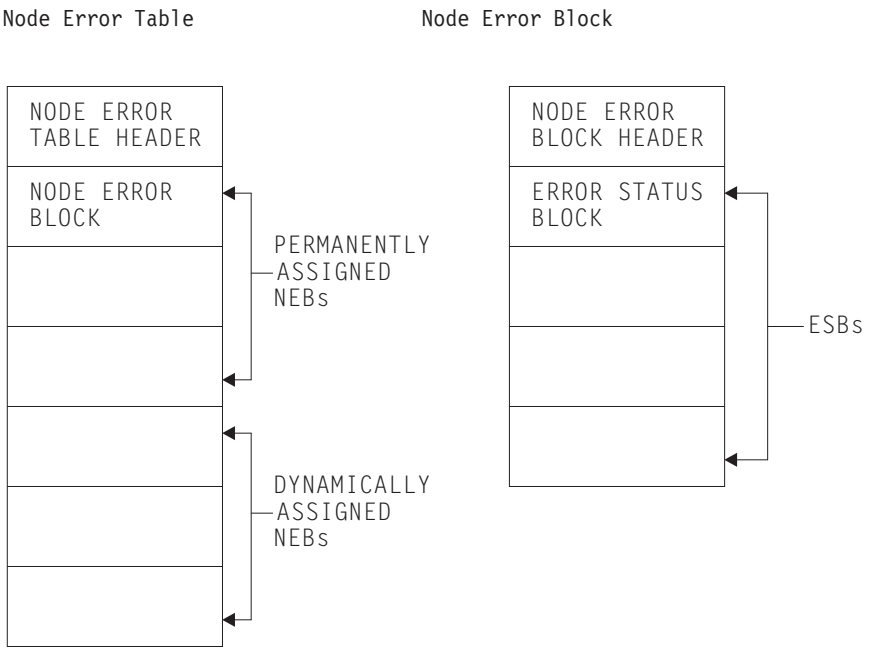


Figure 44. Format of node error table and node error block

Optional common subroutines

The common subroutines are addressed via the CSVT and provide error processors with the following functions:

- Locate or assign NEBs and ESBs on the basis of node ID and error group index.
- Time stamp an error, update an error count, and test an error count against numeric and time threshold values.
- Release a dynamically assigned NEB from a particular node.

Optional error processors for 3270 logical units

Two error processors are supplied for 3270 LUs, as follows:

1. Group index 1, error codes X'D9', X'DC', X'DD', and X'F2'.

   These error codes correspond to the receipt of sense or status bytes in the user sense fields of the RPL. The error processor locates an ESB of the standard format and updates a weighted error count. The weight, threshold, and timer values are based on those used by the sample terminal error program 3270 except as noted in the previous section. If the threshold is not exceeded, the abend SEND, abend RECEIVE, abend transaction flags, and all the print action flags are turned off. Otherwise the default actions are taken and the NEB is released if it is reusable.

2. Group index 2, error code X'42'.

   This code means that no 3270 printer was available to satisfy a print request made at a 3270 screen. The error processor examines the printers defined for this screen to determine why they were unavailable. If either is busy on a previous PRINT or COPY request (that is, a task is attached with a transaction ID of CSPP or CSCY) or is no longer unavailable, that printer address is returned to the node abnormal condition program which retries the print request with an IC PUT command. Otherwise the default actions are taken. (For more details, see the section "Coding for the 3270 'unavailable printer' condition" on page 491.)

Optional error processor for interactive logical units

Oe error processor is supplied for interactive LUs: group index 1, with error code X'DC'.

This error code, in combination with a user sense value of X'081B', indicates a ‘receiver in transmit mode' condition. The action flags in TWANPFW are manipulated to allow the failing SEND request to be retried.

Generating the sample node error program

The routing mechanism, common subroutines, CICS-supplied error processors, and user-supplied error processors are generated by means of DFHSNEP macros.

The sample node error program and table need to be translated, assembled, and link-edited. For information about the job control statements required to assemble and link-edit user-replaceable programs, refer to Chapter 6, “Assembling and link-editing user-replaceable programs,” on page 419.

Note that you should code the translator options NOPROLOG and NOEPILOG in your node error program.

Note also that an extra 24 bytes are required for the common subroutines register save area, and further space is required for the error processor save area. The CICS sample processors use 4 bytes of this area.

The DFHSNEP macro to generate the sample node error program has seven types, as follows:

TYPE=USTOR
to indicate the start of user storage definitions.

TYPE=USTOREND
to indicate the end of user storage definitions.

TYPE=INITIAL
to generate the routing mechanism and, optionally, the common subroutines.

TYPE=DEF3270
to generate the default CICS-supplied error processors for 3270 devices.

TYPE=DEFILU
to generate the default CICS-supplied error processor for interactive logical units operating in contention mode.

TYPE=ERRPROC
to indicate the start of a user-supplied error processor.

TYPE=FINAL
to indicate the end of the sample node error program.

DFHSNEP TYPE=USTOR and USTOREND—defining user storage

The `DFHSNEP TYPE=USTOR` and `DFHSNEP TYPE=USTOREND` macros indicate the start and end respectively of user storage definitions.

The DFHSNEP TYPE=USTOR macro has the following format:

```
DFHSNEP  TYPE=USTOR
```

This macro indicates the start of user storage definitions. It must be followed by your storage definitions, and then by DFHSNEP TYPE=USTOREND. If you use

DFHSNEP TYPE=USTOR to define storage, then both it and DFHSNEP TYPE=USTOREND must be coded **before** DFHSNEP TYPE=INITIAL.

The DFHSNEP TYPE=USTOREND macro has the following format:
This macro indicates the end of user storage definitions. Its use is mandatory if

```
DFHSNEP  TYPE=USTOREND
```

DFHSNEP TYPE=USTOR has been coded. If you use DFHSNEP TYPE=USTOR to define storage, then both it and DFHSNEP TYPE=USTOREND must be coded **before** DFHSNEP TYPE=INITIAL.

DFHSNEP TYPE=INITIAL—generating the routing mechanism

The DFHSNEP TYPE=INITIAL macro indicates the start of the sample node error program and causes the routing mechanism to be generated.

One DFHSNEP TYPE=INITIAL macro must appear immediately **after** DFHSNEP TYPE=USTOR and DFHSNEP TYPE=USTOREND (if they are coded) and **before** the remaining macros.

```
DFHSNEP  TYPE=INITIAL
         [,CS=NO]
         [,NAME=name]
         [,NETNAME=netname]
```

TYPE=INITIAL
indicates the start of the sample node error program and causes the routing mechanism to be generated.

CS=NO
specifies that the generation of the common subroutines is to be suppressed.

NAME=name
specifies the name of the node error program module identifier. The name must be a string of 1 through 8 characters. This operand is optional, and the default is DFHZNEP0. If you allow the NAME operand to default, you can use the examples in Link-edit statements for DFHTEP and DFHZNEP to create link-edit statements, but if you specify a different NAME, you must change the link-edit statements accordingly. If the interface module DFHZNEP (generated by the DFHZNEPI macro) is used, this operand must be specified (with a name other than DFHZNEP).

NETNAME=netname
specifies the name of the node error table to be loaded at initialization. The name must be a string of 1 through 8 characters. This operand is optional, and the default is DFHNET.

DFHSNEP TYPE=DEF3270—including error processors for 3270 LUs

The DFHSNEP TYPE=DEF3270 macro has the following format:

```
DFHSNEP  TYPE=DEF3270
```

TYPE=DEF3270
specifies that the CICS-supplied error processors for 3270 logical units are to be included in the node error program. This macro causes the following source code to be generated:

```
DFHSNEP TYPE=ERRPROC,GROUP=1,CODE=(D9,DC,DD,F2)
Sense/status error processor code.

DFHSNEP TYPE=ERRPROC,GROUP=2,CODE=42
Unavailable printer error processor code.
```

DFHSNEP TYPE=DEFILU—including error processors for INTLUs

The DFHSNEP TYPE=DEFILU macro has the following format:

```
DFHSNEP  TYPE=DEFILU
```

TYPE=DEFILU
specifies that the CICS-supplied error processor for interactive logical units is to be included in the node error program. This macro causes the following source code to be generated:

```
DFHSNEP TYPE=ERRPROC,GROUP=1,CODE=DC
(receiver in transmit mode error processor code)
```

DFHSNEP TYPE=FINAL—terminating DFHSNEP entries

One DFHSNEP TYPE=FINAL macro must follow all the other DFHSNEP macros to indicate the end of the node error program.

It has the following format:

```
DFHSNEP  TYPE=FINAL
```

TYPE=FINAL
indicates the end of the node error program and causes the error processor vector table (EPVT) to be generated. The EPVT is a table containing addresses of the error group processors invoked by the routing mechanism of the node error program.

DFHSNEP TYPE=ERRPROC—specifying a user error processor

The DFHSNEP TYPE=ERRPROC macro is used to indicate the start of a user-supplied error processor. The actual error processor code should immediately follow this macro. The assembly should be terminated by the statement `END DFHNEPNA`.

The following operands can be used on the DFHSNEP TYPE=ERRPROC macro:

```
DFHSNEP  TYPE=ERRPROC
         ,CODE=(error-code,...)
         ,GROUP=error-group-index
```

TYPE=ERRPROC
indicates the start of a user-supplied error processor.

CODE=(error-code,...)
specifies the error codes that make up the error group, and which are therefore handled by the error processor supplied. The operand is coded as a sublist of 2-character representations of 1-byte hexadecimal codes. (The parentheses can be omitted for a single code.) For each code specified, the error group index is placed at the equivalent offset in the translate table. Thus, when this code occurs, the appropriate error processor can be identified.

GROUP=error-group-index
specifies an error group index for the error processor. This index is used to name the error processor, locate its address from the error processor vector table (EPVT), and optionally associate it with an ESB in each NEB. The index

specified must be a 2-character representation of a 1-byte hexadecimal number in the range X'01' through X'FF' (a leading zero can be omitted). The error processor name has the form NEPROCxx, where "xx" is the error group index. A CSECT statement of this name is generated, which causes the error processor code to be assembled at the end of the node error program module and to have its own addressability.

If you intend to add your own error processors to the sample node error program, you should consider the following factors:

- The layout of the communication area. The communication area is described in detail in Figure 41 on page 474.
- The fact that certain functions cannot be used within DFHZNEP. (See "Restrictions on the use of **EXEC CICS** commands" on page 490.)
- The register conventions used by the sample node error program. These are described in Table 25.

Table 25. Register assignment

Register	Use
0	Work register
1	Address of the EXEC parameter list
2	NEB base register (DFHSNEP only)
3	ESB base register (DFHSNEP only) NEP error class register (DFHZNEPI only)
4	NEP name pointer register (DFHZNEPI only)
5	NEP interface base register (DFHZNEPI only)
6	Work register
7	Work register
8	Work register
9	Work register
10	Code base register
11	Address of the EIB
12	Address of the communication area
13	Address of DFHEISTG storage
14	CSVT base and error processor link register Common subroutine link register
15	Error processor branch register Common subroutine branch register.

Note:

1. Register 14 must be saved for return from error processors. The common subroutine vector table (CSVT) is coded after the BALR to the error processor and so this register is also the CSVT base.
2. Registers 1, 10, 12, 13, 14, and 15 are set up on entry to error processors.
3. Registers 14 through 11 can be saved by error processors in an area reserved in EXEC interface storage at label NEPEPRS. Registers 15 through 11 do not need to be restored before return from error processors.
4. Registers 4 through 9 can be saved by common subroutines in an area reserved in EXEC interface storage at label NEPCSRS. They must be restored before return from the subroutines.

DFHSNET—generating the node error table

The DFHSNET macro is used to generate a node error table. Each node error table that you generate must be defined to CICS.

```
DFHSNET  [NAME=DFHNET|name]
         [,COUNT=100|threshold]
         [,ESBS=1|(index,length,...)]
         [,NEBNAME=(name,...)]
         [,NEBS=10|number]
         [,TIME=(7,MIN)|(interval,units)]
```

NAME=DFHNET|name
specifies the identifier to be included in the NET header. It must be a string of one through eight characters. This operand is optional, and the default is DFHNET.

COUNT=100|threshold
specifies the error count threshold that is to be stored in the NET header for use by the common subroutines to update standard ESBs. If the threshold is exceeded, the error processor that invoked the subroutine is informed by a return code. The maximum value is 32 767. This operand is optional, and the default is 100.

ESBS=1|(index,length,...)
specifies the ESB structure for each NEB. This operand is coded as a sublist. Each element of the sublist comprises two values: "index" specifies an error group index for which an ESB is to be included in the NEB; "length" specifies the status area length, in bytes, for that ESB. The parentheses can be omitted for a single element. The "index" must be specified as a 2-character representation of a 1-byte hexadecimal number in the range X'01' through X'FF' (a leading 0 can be omitted). The "length" is constrained only because an 8-byte NEB header plus a 4-byte header for each ESB must be contained within the maximum NEB length of 32 767 bytes. If a null value is specified, a standard ESB with a status area length of 10 bytes is assumed. This is suitable for use by the common subroutines in maintaining a time-stamped error count.

This operand is optional and defaults to 1. This causes each NEB to be generated with one ESB for error group 1 with a status area length of 6 bytes.

NEBNAME=(name,...)
specifies the names of nodes that are to have a permanently assigned NEB. The names specified are assigned, in the order specified, to the set of NEBs requested by the NEBS operand. Any remaining NEBs are available for dynamic allocation to other nodes as errors occur. The name must be a string of 1 through 4 characters. The parentheses can be omitted for a single name. This operand is optional and has no default.

NEBS=10|number
specifies the number of NEBs required in the NET. The maximum valid number is 32 767; the default is 10.

TIME=(7,MIN)|(interval,units)
specifies the time interval that is to be stored in the NET header for use by the common subroutines to maintain error counts in standard ESBs. If the threshold specified in the COUNT operand is not exceeded before this time interval elapses, the error count is reset to 0. Specify "units" as SEC, MIN, or HRS. The maximum values for "interval" are as follows: (86400,SEC), (1440,MIN), or (24,HRS). This operand is optional, and the default is set to (7,MIN).

Node error program DSECTs

CICS provides a number of DSECTs for use in the node error program (NEP).

The following DSECTs are provided:

Node Error Table Header: This contains the table name and common information relevant for all the node error blocks (NEBs) in the table.

```
DFHNETH    DSECT
NETHNAM    DS      CL8         Table name
NETHNBN    DS      H           Number of NEBs in table
NETHNBL    DS      H           Length of NEBs in table
NETHTIM    DS      PL8         Error count time interval
NETHECT    DS      H           Error count threshold
NETHFLG    DS      X           Flag byte
NETHINI    EQU     X'01'       Table initialized
           DS      X           Reserved
NETHFNB    DS      0F          First NEB
```

Node Error Block: The table contains node error blocks that are used for recording error information for individual nodes. These can be permanently assigned to specific nodes or dynamically assigned at the request of error processors.

```
DFHNETB    DSECT
NEBNAM     DS      CL4         Node name
NEBFLG     DS      X           Flag byte
NEBPERM    EQU     X'01'       Permanently assigned NEB
           DS      XL3         Reserved
NEBFESB    DS      0X          First NEB
```

Error Status Block: The NEBs can contain error status blocks. These are reserved for specific error processors and are identified by the corresponding error group index. An ESB can have a format defined by you, or can have a standard format suitable for counting errors over a fixed time interval.

```
DFHNETE    DSECT
ESBEGI     DS      X           Error group index
ESBFLG     DS      X           Flag byte
ESBSTAN    EQU     X'01'       Standard format ESB
ESBTTE     EQU     X'02'       Time threshold exceeded
ESBCTE     EQU     X'04'       Count threshold exceeded
ESBSLEN    DS      XL2         Status area length
ESBHLEN    EQU     *-DFHNETE   ESB header length
ESBSTAT    DS      0X          Status area
```

The following fields apply to the standard format:

```
ESBTIM     DS      PL8         Time stamp
ESBEC      DS      XL2         Error count
```

Common Subroutine Vector Table: The CSVT provides error processors with addressability to the common subroutines. The error processor link register gives addressability to the CSVT and so the first section of the DSECT overlies the code required to branch around the actual table.

```
DFHNEPC    DSECT
           DS      F         Load instruction
           DS      F         Branch instruction
CSVTNEP    DS      A         Node error program base address
CSVTESBL   DS      A         NEPESBL - ESB locate routine
CSVTNEBD   DS      A         NEPNEBD - NEB delete routine
CSVTECUP   DS      A         NEPECUP - error count update
                             routine
```

Writing your own node error program

You can write your own node error program (NEP) in any of the CICS-supported languages.

CICS provides NEP code is provided in assembler language, and the communication area parameter list is supplied in assembler language and C versions. The names of the supplied source files, copy books, and macros, and the libraries in which they can be found, are listed in Table 26.

Table 26. Supplied source files, copy books, and macros

Name	Type	Description	Library
DFHZNEP0	Program	Default node error program (assembler language)	CICSTS52.CICS.SDFHSAMP
DFHZNEPX	Source	Default NEP (embedded by DFHZNEP0 via COPY statement)	CICSTS52.CICS.SDFHSAMP
DFHSNEP	Macro	Sample NEP program generator (assembler language)	CICSTS52.CICS.SDFHMAC
DFHZNEPI	Macro	NEP interface generator (for multiple NEPs)	CICSTS52.CICS.SDFHMAC
DFHNEPCA	Macro	assembler language communication area	CICSTS52.CICS.SDFHMAC
DFHNEPCA	Copy book	C-language communication area	CICSTS52.CICS.SDFHC370

If you code in assembler language, you can use the sample NEP as a framework on which to construct your own node error program.

Restrictions on the use of `EXEC CICS` commands

The commands that a node error program (NEP) can issue are restricted. In particular, do not use commands that require a principal facility, because their results are unpredictable.

Do not use commands that start the following functions:

- Terminal control. For example, issuing an **`EXEC CICS DELAY`** command can cause the CSNE task to suspend and never resume, which can cause shutdown of the region to hang. CEMT-type commands, however, such as **`EXEC CICS INQUIRE TERMINAL`**, are permitted.
- BMS (except routing).
- ISC communication (including function shipping), including **`START`** requests for remote transactions, although such requests are not recommended because CSNE (Node Abnormal Condition task) might become suspended while issuing an ALLOCATE command to the remote system.

  To start a remote transaction, start a local transaction which in turn starts a remote transaction.
- Updates to recoverable resources. If the resources are locked by another task, the CSNE unit of work can be suspended or shunted.

You cannot use the NEP to suppress DFHZNAC messages.

Entry and addressability

On entry, your NEP should issue the commands:

```
EXEC CICS ADDRESS COMMAREA
EXEC CICS ADDRESS EIB
```

These commands provide addressability to the communication area passed by DFHZNAC, and to the EXEC interface block, respectively.

If you write your node error program in assembler language, you generate the communication area DSECT by coding:

```
DFHNEPCA TYPE=DSECT
```

If you write your program in C, you include the communication area definitions by coding:

```
#include <dfhnepca>
```

Coding for the 3270 ‘unavailable printer' condition

About this task

The ‘unavailable printer' condition arises when a print request is made using the 3270 print request facility, and there are no printers on the control unit, or when the printers are in one of the following conditions:

- Out of service
- Not in TRANSCEIVE or RECEIVE status for automatic transaction initiation
- With a task currently attached
- Busy on a previous operation
- Requiring intervention.

The procedure is applicable to 3270 logical units or to the 3270 compatibility mode logical unit when using the PRINTER and ALTPRINTER operands of the CEDA DEFINE TERMINAL command.

The terminal control program recognizes this condition, and issues a READ BUFFER request to collect the data into a terminal I/O area. The TIOA is of the same format as it is when an application program has issued a terminal control read buffer request.

The terminal control program z/OS Communications Server section (DFHZCP) then queues the TCTTE to the node abnormal condition program with error code X'42' (TCZCUNPRT). The node abnormal condition program (DFHZNAC) writes to the CSNE transient data queue:

- DFHZC2497 UNAVAILABLE PRINTER (device types 3270P and LUTYPE3)
- DFHZC3493 INVALID DEVICE TYPE FOR A PRINT REQUEST (all other printer device types).

Before linking to the node error program, DFHZNAC inserts the primary and secondary printer netnames and terminal IDs into the communication area, indicating also whether either printer is eligible for a print request. DFHZNAC links to the node error program with no default actions set.

On return from the node error program, DFHZNAC checks the additional system parameter TWAUPRRC in the communication area (see Figure 41 on page 474) and, based on its contents, performs one of the following actions:

- If your NEP sets TWAUPRRC to X'FF' (-1), DFHZNAC assumes that the node error program has disposed of the data to be printed and therefore takes no further action.
- If your NEP sets TWAUPRRC to zero, DFHZNAC assumes that no printer is available and takes no further action.
- If your NEP sets TWAUPRRC to neither zero nor -1, DFHZNAC assumes that one of either field TWAPNETN or field TWAPNTID is set. (If both are set, TWAPNTID(termid) takes precedence.) An interval control PUT is performed to the provided terminal. The transaction to be initiated is CSPP (print program), and the time interval is zero.
  - If an error occurs on the interval control PUT, DFHZNAC writes the 'DFHZC2496 IC FAILURE' message to the destination CSNE. DFHZNAC then links to the node error program again with the TWAUPRRC field set to -2. This is done to give the node error program a last chance to dispose of the data. On the second return from the node error program to DFHZNAC, the latter reexamines TWAUPRRC. If TWAUPRRC is -1, then the node error program has disposed of the data.
  - If no error occurs on the interval control PUT, DFHZNAC checks for the following printer conditions:
    - 'Out of service'
    - 'Intervention required'
    - Any condition other than RECEIVE or TRANSCEIVE status.

    If one of these conditions is true, DFHZNAC issues the 'DFH2495 PRINTER OUTSERV/IR/INELIGIBLE-REQ QUEUED' message to the destination CSNE.

Finally, DFHZNAC terminates any print requests on the originating terminal and performs normal action flag processing on that terminal.

Coding for session failures

Following some categories of error associated with logical unit or path failures, the session between CICS and the logical unit may be lost. The default action taken by DFHZNAC may be to put the TCTTE out of service.

About this task

A method of automatically reacquiring the session is for your node error program to alter the default DFHZNAC actions and to keep the TCTTE in service. Your node error program can then issue an EXEC CICS START TERMID(name) command against that TCTTE for a transaction written in a similar manner to the CICS "good morning" signon message (CSGM). When the transaction is initiated using automatic transaction initiation (ATI), CICS tries to reacquire the session. If the session fails again, DFHZNAC is reinvoked and the process is repeated.

The time specified on the EXEC CICS START command is determined by installation-dependent expected-mean-time-to values.

If used in this way, the initiated transaction can write an appropriate signon message when the session has been acquired. Note, however, that if LOGONMSG=YES is specified on the CEDA DEFINE TYPETERM command, the CICS "good morning" message is also initiated when the session is opened. Refer to "Restrictions on the use of **EXEC CICS** commands" on page 490.

Coding for specific SNA sense codes

About this task

Figure 45 shows how your NEP error processors could test for the presence of specific SNA sense codes.

```
TEST1   EQU   *
        CLC   TWASENSR(2),SNS1              SENSE CODE EQUAL TO NNNN
        BNE   TEST2                         NO, THEN NEXT TEST
        NI    TWAOPT1,TWAOAF                PRINT ACTION MESSAGES ONLY
        OI    TWAOPT2,TWAOAS+TWAOAR+TWAOAT  ABANDON SEND,RECEIVE AND TASK
        NI    TWAOPT2,255-TWAOASM           SET SIMLOGON OFF
        OI    TWAOPT3,TWAOINT               SET INTLOG NOW ALLOWED
        NI    TWAOPT3,255-TWAONINT          OR RESET NOINTLOG
        B     END
        .
        .
        .
SNS1    DC    X'NNNN'
```

Figure 45. Sample code, showing how your node error program could test for specific SNA sense codes

Writing multiple NEPs

You can write several node error programs, as described in "Multiple NEPs" on page 471. When an error occurs, the node abnormal condition program passes control to an interface module, DFHZNEPI, which determines the transaction class and passes control to the appropriate node error program.

If only one node error program is used, the interface module (DFHZNEPI) is not required. If the node error program is named DFHZNEP, the node abnormal condition program branches directly to that. If more than one node error program is used, the interface module (DFHZNEPI) is required. In this case, the node error programs must be given names other than DFHZNEP. There must be an installed program definition for every node error program generated.

DFHZNEPI macros

The following macros are required to generate the node error program interface module (DFHZNEPI):

- DFHZNEPI TYPE=INITIAL to specify the name of the default transaction-class routine.
- DFHZNEPI TYPE=ENTRY to associate a transaction-class with a transaction-class error handling routine.
- DFHZNEPI TYPE=FINAL to end the DFHZNEPI entries.

The DFHZNEPI interface module must be generated when you require the node abnormal condition program to pass control to the appropriate user-written node error program for resolution of the error.

DFHZNEPI TYPE=INITIAL—specifying the default routine

The DFHZNEPI TYPE=INITIAL macro specifies the name of the default transaction-class routine to be used for the DFNZNEPI module.

Syntax

```
DFHZNEPI  TYPE=INITIAL
          [,DEFAULT=name]
```

DEFAULT=name
specifies the name of the default transaction-class routine to be used. A link is made to this default routine if you specify for the transaction (using the CEDA DEFINE PROFILE, CEDA DEFINE SESSIONS, or CEDA DEFINE TYPETERM command) a NEPCLASS value of 0 (the default) or higher than 255, or if you do not specify a transaction-class routine using the DFHZNEPI TYPE=ENTRY macro for the class specified on the NEPCLASS operand.

If either of the preceding conditions is true, but you do not code the DEFAULT operand, then no routine is invoked.

The DFHZNEPI TYPE=INITIAL macro must always be specified, and must be placed before any other forms of the DFHZNEPI macro. Only one TYPE=INITIAL macro can be specified.

DFHZNEPI TYPE=ENTRY—specifying a transaction-class routine

Use the DFHZNEPI TYPE=ENTRY macro to associate a transaction class (NEPCLASS) with a transaction-class error handling routine.

The format of this macro is:

```
DFHZNEPI  TYPE=ENTRY
          ,NEPCLAS=integer
          ,NEPNAME=name
```

NEPCLAS=integer
specifies the transaction-class, and must be in the range 1 through 255. No value should be specified that has been specified in a previous DFHZNEPI TYPE=ENTRY instruction.

NEPNAME=name
specifies a name for the transaction-class routine to be associated with the specified transaction-class. An error condition results if the name is specified either as DFHZNEP, or is longer than 8 characters.

Note: You can use the sample node error program (with a name other than DFHZNEP) as a transaction-class routine for the interface module, DFHZNEPI.

DFHZNEPI TYPE=FINAL—terminating DFHZNEPI entries

```
DFHZNEPI  TYPE=FINAL
```

TYPE=FINAL
completes the definition of module DFHZNEPI and must be specified last. The assembly should be terminated by an END statement with no entry name specified, or by the statement: END DFHZNENA.

Handling shutdown hung terminals in the node error program

For error code X'6F', DFHZNAC passes the setting of the **TCSACTN** system initialization parameter and the DFHZC2351 reason code to DFHZNEP. DFHZNEP can modify the force-close action for the current terminal.

Error Code
X'6F'

Symbolic Name
TCZSDAS

Message Number
DFHZC2351

The setting of TCSACTN is passed to DFHZNEP by setting TWAOSCN:

- TWAOSCN off (B'0') indicates TCSACTN=NONE.
- TWAOSCN on (B'1') indicates TCSACTN=UNBIND.

The DFHZC2351 reason code is passed to DFHZNEP in the NEP communications area (NEPCA) field TWATRSN. TWATRSN is a 1-byte code field. Currently, TWATRSN overlays TWAREASN (also a 1-byte field). The codes, and their meaning, are as follows:

01 Request in progress

02 Task still active

03 Waiting for SHUTC

04 Waiting for BIS

05 Waiting for UNBIND

06 Waiting for RTR

07 BID in progress

08 Other TC work pending

99 (X'63') Undetermined

For further details, see the terminal control message DFHZC2351.

DFHZNEP can modify the force-close action for the current terminal by setting TWAOSCN:

- If DFHZNEP sets TWAOSCN off (B'0'), DFHZNAC does not attempt to force-close the terminal (TCSACTN=NONE).
- If DFHZNEP sets TWAOSCN on (B'1'), DFHZNAC attempts to force-close the terminal (TCSACTN=UNBIND). Internally, DFHZNAC achieves this by converting the TWAOSCN normal close to a TWAOCN forced close.

DFHZNEP cannot modify the TCSWAIT or TCSACTN system initialization parameters (see TCSWAIT system initialization parameter in Reference -> System definition and TCSACTN system initialization parameter in Reference -> System definition).

Using the node error program with persistent sessions

This section contains guidance information about the NEP in a persistent sessions environment.

The node error program with persistent session support

Persistent session support is described in the CICS Recovery and Restart Guide.

One of the steps in the conversation-restart process is to link to the node error program with error code X'FD'. If you want to be able to change any of the system-wide recovery notification options (whether terminal users are notified of a recovery, the recovery message, or the recovery transaction) for some terminals, you should write your own error processor to handle code X'FD'.

When using persistent sessions, note the following:

- When a session has been recovered, it may be a good idea to run NEP processing equivalent to your normal “session started” (code X'48') processing, because code X'48' is not passed on session recovery when persistent sessions are used.
- In certain situations where sessions have persisted over a failure but have been unbound on restart (for example, a COLD start occurs after a CICS failure), the NEP is not driven. (In systems without persistent sessions support, the NEP is always driven with code X'49', “session terminated”, when a z/OS Communications Server session terminates.) Conditions leading to the issuing of the following messages do not drive the NEP. The messages appear on the system console:

  ```
  DFHZC0120     DFHZC0124
  DFHZC0121     DFHZC0129
  DFHZC0122     DFHZC0130
  DFHZC0123
  ```

  Conditions leading to the issuing of messages `DFHZC0125` and `DFHZC0131` drive the NEP with codes X'FB' and X'FC' respectively. It is recommended that you run NEP processing equivalent to your normal “session terminated” NEP processing for these conditions.
- If zero is specified on the AIRDELAY system initialization parameter, autoinstalled terminals are not recovered after a restart. Similarly, if the delay period specified on AIRDELAY expires before an autoinstalled terminal is used after a restart, the terminal definition is deleted. In these circumstances, any expected NEP processing as a result of CLSDSTP=NOTIFY being coded does not take place.

Changing the recovery notification

The method of recovery notification for a terminal is defined using the RECOVNOTIFY option of the TYPETERM definition, which is described in the *CICS Resource Definition Guide*. This is the most efficient way to specify the recovery notification method for the whole network, because CICS initiates the notification procedure during the early stages of takeover.

You can use the node error program to change the recovery notification method for some of the switched terminals. For example, you may want most terminals of a particular type to receive the recovery message at takeover, but the rest to get no notification that service has been restored. To achieve this, you could code RECOVNOTIFY(MESSAGE) in the TYPETERM definition, and then use the node error program to change the recovery notification to NONE for the relatively few terminals that are not to be notified.

Changing the recovery message

If you define a terminal with RECOVNOTIFY(MESSAGE) in its TYPETERM definition, a recovery message is sent to the terminal after takeover.

By default, for an XRF takeover, the message is the following CICS-supplied message which is defined in BMS map DFHXRC1 of map set DFHXMSG:

```
CICS has recovered after a system failure.
Execute recovery procedures.
```

For a persistent sessions recovery, BMS map DFHXRC3 is used; this map prefixes the above message with CICS message number DFHZC0199. You can specify your own map set in the node error program if you want to change the recovery

message for some of the switched terminals. This could be useful, for example, if signon security is in force and you want to tell terminal users to sign on again. The map set that you specify must have an installed program definition. If you choose to change the recovery message for all terminals, it would be more efficient to replace the CICS-supplied map with your own.

Changing the recovery transaction

The recovery transaction, to be started at a terminal after takeover, is specified using the RMTRAN system initialization parameter. This is the most efficient way of specifying a recovery transaction for the network. You can specify a different transaction for some of the switched terminals by overwriting field TWAXTRAN in the communication area. The transaction that you specify must have an installed transaction definition, and the terminal must be defined with the option ATI(YES).

If the transaction specified in TWAXTRAN does not exist, CICS tries to start the CSGM transaction. If this also fails, CICS terminates the session.

Using the node error program with z/OS Communications Server generic resources

The **EXEC CICS ISSUE PASS** command can be used (either from an application program or CECI) to disconnect a LU from CICS, and transfer it to the z/OS Communications Server application specified on the LUNAME option. For example, to transfer a LU from this CICS to another LU-owning region, you could issue the command:

```
CECI ISSUE PASS LUNAME(applid)
```

where `applid` is the APPLID of the TOR to which the LU is to be transferred.

If your TORs are members of a z/OS Communications Server generic resource group, you can transfer a LU to any member of the group by specifying LUNAME as the generic resource name. For example:

```
CECI ISSUE PASS LUNAME(grname)
```

where `grname` is the generic resource name. z/OS Communications Server chooses the most suitable group member to which to transfer the LU. (If you need to transfer a LU to a specific TOR within the CICS generic resource group, you must specify LUNAME as the member name—that is, the CICS APPLID, as in the first example.)

Note that, if the system that issues an ISSUE PASS LUNAME(grname) command is the *only* CICS currently registered under the generic resource name (for example, the others have all been shut down), the ISSUE PASS command does **not** fail with an INVREQ. Instead, the LU is logged off and message DFHZC3490 is written to the CSNE log.

You may want to code your node error program to deal with the situation when message DFHZC3490 (DFHZNAC error code X'C3') is issued.

Chapter 12. Writing a program to control autoinstall of LUs

You can write a program to control the automatic installation of locally-attached SNA LUs, including APPC single-session devices.

For information about controlling the automatic installation of local APPC connections that are initiated by BIND requests, see Chapter 14, "Writing a program to control autoinstall of APPC connections," on page 527. For information about controlling the installation of shipped LUs and connections, see Chapter 16, "Writing a program to control autoinstall of shipped terminals," on page 545. For information about controlling the installation of virtual LUs used by the CICS Client products and the 3270 bridge , see Chapter 17, "Writing a program to control autoinstall of virtual terminals," on page 553.

Autoinstalling terminals

You use the **DEFINE TERMINAL** and **DEFINE TYPETERM** commands to define z/OS Communications Server devices to CICS. These commands define the resource definitions in the CICS system definition file (CSD). Your definitions can specify that they can be used as models for autoinstall purposes.

Defining and installing model resource definitions for terminal control enables CICS to create required entries in the terminal control table (TCT) automatically, whenever unknown devices request connection to CICS. A particular advantage of automatic installation (autoinstall) is that the resource occupies storage in the TCT only while it is connected to CICS and for a specified delay period after last use.

You use the autoinstall control program to select some of the data needed to automatically install your terminals—notably the CICS terminal name and the model name to be used in each instance. You can use the CICS-supplied autoinstall program, or extend it to suit your own purposes.

For an overview of autoinstall, see the *CICS Resource Definition Guide*. You should also read the sections in the same manual that describe the CEDA commands that create the environment in which your control program can work.

If you choose automatic installation for some or all of your terminals, you must:

1. Create some model TERMINAL definitions.
2. Define the terminals to the z/OS Communications Server, so that their definitions in the z/OS Communications Server match the model TERMINAL definitions in CICS.
3. If you are using model terminal support (MTS), define the MTS tables to the z/OS Communications Server.
4. Use the default autoinstall control program for terminals (DFHZATDX), or write your own program, using the source-code of the default program and the customization examples in these topics as a basis. (You can write an entirely new program if the default program does not meet your needs, but you are recommended to try a default-based program first.) You can write your program in any of the languages supported by CICS - the source of the default program is provided in assembler language, COBOL, PL/I, and C. You can rename your user-written program.

Note:

a. You must compile your autoinstall control program (or the supplied DFHZCTDX) using a Language Environment - enabled compiler, and you must run the program with Language Environment enabled.

b. You can have only one active autoinstall control program to handle both terminals and APPC connections. You specify the name of the active program on the **AIEXIT** system initialization parameter. The DFHZATDY program described in Chapter 14, "Writing a program to control autoinstall of APPC connections," on page 527 provides the same function for terminal autoinstall as DFHZATDX, but in addition provides function to autoinstall APPC connections initiated by BIND requests. Both DFHZATDX and DFHZATDY provide function to install shipped terminals and connections. So, for example, if you want to autoinstall APPC connections as well as SNA LUs, you should use a customized version of DFHZATDY, rather than DFHZATDX.

Coding entries in the z/OS Communications Server LOGON mode table

CICS uses the logmode data in the z/OS Communications Server LOGON mode table when processing an autoinstall request. Autoinstall functions properly only if the logmode entries that you define to z/OS Communications Server have matches among the model TERMINAL definitions that you specify to CICS.

The tables in Appendix B, "Coding entries in the z/OS Communications Server LOGON mode table," on page 873 show, for a variety of possible LU devices, what you must have coded on the z/OS Communications Server MODEENT macros that define, in your logmode table, the LUs that you want to install automatically. Between them, the tables show the values that must be specified for each of the operands of the MODEENT macro.

Some of the examples in the appendix correspond exactly to entries in the IBM-supplied logon mode table called ISTINCLM. Where this is so, the table gives the name of the entry in ISTINCLM.

Using model terminal support (MTS)

Using MTS, you can define the model name, the printer (PRINTER), and the alternate printer (ALTPRINTER) for each LU in an SNA table.

This information is sent by z/OS Communications Server in an extended CINIT RU. CICS captures it as part of autoinstall processing at logon, and uses it to create a TCTTE for the LU.

Coding entries for MTS

You must define model names (MDLTAB, MDLENT, and MDLPLU macros) and printer and associated printer names (ASLTAB, ASLENT, and ASLPLU macros) to z/OS Communications Server.

The autoinstall control program for terminals

In addition to managing your resource definition, your autoinstall control program can perform any other processes that you want at this time. Its access to the command-level interface is that of a normal, nonterminal user task.

Some possible uses are listed “Sample autoinstall control programs for terminals” on page 511.

The control program is invoked when:

1. An autoinstall INSTALL request is being processed
2. An autoinstall DELETE request has just been completed
3. An autoinstall request has previously been accepted by the user program, but the subsequent INSTALL process has failed.

On each invocation of the autoinstall control program, a parameter list is passed (using a communication area), describing the function being performed (INSTALL or DELETE), and providing data relevant to the particular event. (In case 3, the control program is invoked as if for DELETE).

The INSTALL and DELETE events are now described in detail.

The autoinstall control program at INSTALL

If autoinstall is operative, the autoinstall control program is invoked at INSTALL for:

- Local SNA LUs
- MVS consoles
- Local APPC single-session connections initiated by a CINIT
- Local APPC parallel-session connections initiated by a BIND
- Local APPC single-session connections initiated by a BIND
- Client virtual terminals
- Shipped terminals and connections.

On each invocation, CICS passes a parameter list to the control program by means of a communication area addressed by DFHEICAP. The parameter list passed at INSTALL of MVS consoles is described in “Autoinstall control program at INSTALL” on page 520. The parameter list passed at INSTALL of local APPC connections initiated by BIND requests is described in “The communication area at INSTALL for APPC connections” on page 529. The parameter list passed at INSTALL of shipped terminals and connections is described in “The communications area at INSTALL for shipped terminals” on page 547. The parameter list passed at INSTALL of client virtual terminals is described in “The communications area at INSTALL for Client virtual terminals” on page 557. The parameter list passed at INSTALL of MVS consoles is described in Chapter 13, “Writing a program to control autoinstall of consoles,” on page 519. This section describes only INSTALL of local terminals (including APPC single-session connections initiated by a CINIT).

The control program is invoked at INSTALL for terminals when both:

- A z/OS Communications Server for SNA logon request has been received from a resource eligible for automatic installation whose NETNAME is not in the TCT.
- Autoinstall processing has been completed to a point where information (a terminal identifier and autoinstall model name) from the control program is required to proceed.

The communication area at INSTALL for terminals

The layout of the communication area is shown in Figure 46.

```
Fullword 1            Standard Header
  Byte  1             Function Code          (X'F0' for INSTALL)
  Bytes 2 - 3         Component Code         Always 'ZC'
  Byte  4             Reserved               Always X'00'
Fullword 2            Pointer to NETNAME_FIELD
Fullword 3            Pointer to MODELNAME_LIST
Fullword 4            Pointer to SELECTED_PARMS
Fullword 5            Pointer to CINIT_RU
```

Figure 46. Autoinstall control program's communication area at INSTALL

The parameter list contains the following information:

1. Standard Header. Byte 1 indicates the request type (this is hexadecimal character X'F0' for INSTALL).
2. Pointer to a 2-byte length field, followed by the NETNAME of the resource requesting LOGON.
3. Pointer to an array of names of eligible autoinstall models. The array is preceded by a 2-byte field describing the number of 8-byte name elements in the array. If there are no elements in the array, the number field is set to zero.
4. Pointer to the area of storage that you use to return information to CICS, and where the MTS information from the z/OS Communications Server CINIT is stored.
5. Pointer to z/OS Communications Server LOGON data (the CINIT request unit). The data is preceded by a 2-byte length field, indicating the length of the CINIT request unit, and includes the 3-character NS header. The format of the CINIT request unit is described in the *SNA Network Product Formats* manual.

CICS passes a list of eligible autoinstall models in the area addressed by fullword 3 of the parameter list.

If the model name is not supplied by MTS, the control program must select a model from this list that is suitable for the device logging on, and move the model name to the first 8 bytes of the area addressed by fullword 4 of the parameter list.

For example, if a 3270 printer attempts to autoinstall, the subset of matching models includes all the types in z/OS Communications Server category 2 that you have defined as models. This subset could include any of the following:

- DEVICE(3270) TERMMODEL(2)
- DEVICE(3270) TERMMODEL(1)
- DEVICE(3270P) TERMMODEL(2)
- DEVICE(3270P) TERMMODEL(1)
- DEVICE(3275) TERMMODEL(2)
- DEVICE(3275) TERMMODEL(1).

The control program selects one model from this list, and CICS uses this model to build the TCTTE for the device. The default autoinstall control program, DFHZATDX, always selects the first model name in the list.

If you are not using MTS but need a printer ID or NETNAME (or an alternative printer ID or NETNAME) associated with this terminal, then your control program can supply this in the area addressed by fullword 4.

If you are using MTS, CICS passes the control program the printer and alternative printer NETNAMEs specified on the z/OS Communications Server ASLTAB macro.

Before returning to CICS, the control program must supply a CICS terminal name for the device logging on, and must set the return code field to X'00' if the autoinstall request is to be allowed.

Figure 47 shows all of these fields in their required order.

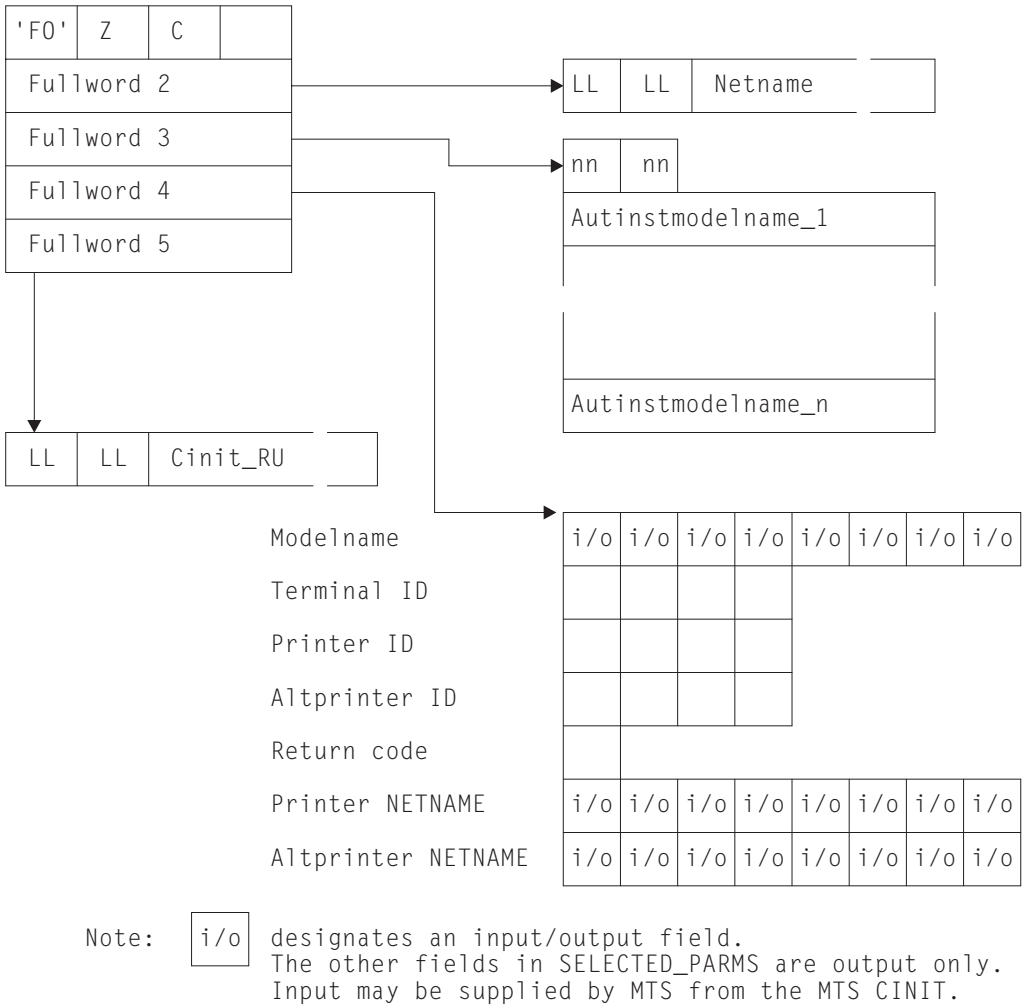


Figure 47. Autoinstall control program's parameter list at INSTALL

How CICS builds the list of autoinstall models

If CICS finds an MTS model name (and the model is defined to CICS and is compatible with the z/OS Communications Server information describing the resource), CICS puts the model name into the model name list (Autinstmodelname_1), and also into the model name field (Modelname) in the selection list addressed by fullword 4 of the parameter list.

If CICS is unable to find an MTS model name in the MTS Control Vector, or the named model does not exist or is invalid, it builds the list of autoinstall models by selecting from the complete list of terminal models those models that are compatible with the z/OS Communications Server information describing the resource. The complete list of autoinstall models available to CICS at any time comprises all the definitions with AUTINSTMODEL(YES) and AUTINSTMODEL(ONLY) that have been installed, both by the GRPLIST at a CICS initial or cold start, and by INSTALL GROUP commands issued by CEDA. The *CICS Resource Definition Guide* describes the definition of models.

Table 48 on page 874 gives you the information to work out which model types could be included in the subset of models passed to the autoinstall control program when a particular terminal attempts to install. The subset is determined by the z/OS Communications Server characteristics of the device attempting to log on. The number in the right-hand column of the figure indicates the selection of the subset from the full list. When a terminal with a given combination of DEVICE, SESSIONTYPE, and TERMMODEL values attempts to logon, the subset of matching models passed to the control program includes all the models with DEVICE, SESSIONTYPE, and TERMMODEL values that have a corresponding z/OS Communications Server category number in the right-hand column of the table.

If CICS finds no model that exactly matches the BIND, and if the return code in the area addressed by fullword 4 of the parameter list is nonzero, then CICS issues error message DFHZC6987. This message contains a "best failure" model name, which is provided for diagnostic purposes only. It is described in detail in "CICS action on return from the control program" on page 507, and in the *CICS Messages and Codes Vol 1* manual.

Returning information to CICS

At the INSTALL event, the autoinstall control program is responsible for allowing or denying the connection of a new terminal resource to the CICS system. This decision can be based on a number of installation-dependent factors, such as security or the total number of connected terminals. CICS takes no part in any such checking. You decide whether any such checking takes place, and how it is done.

If the INSTALL request is to proceed, the control program must do the following:

- Return an autoinstall model name in the first 8 bytes of the area addressed by fullword 4 of the parameter list, unless this is already set by MTS support.

  If the control program returns a model name that is not in the subset passed to it by CICS, CICS cannot guarantee what will happen when further processing takes place. It is the user's responsibility to determine the effect of associating any particular logon request with a particular model name, because no interface is provided to the in-storage "model" objects.
- Supply a CICS terminal name (TERMID) in the next four bytes of the return area.

  DFHZATDX takes the last four nonblank characters of the NETNAME (addressed by fullword 2 of the parameter list) as the terminal name, so you must code your own autoinstall program if this does not match the naming conventions of your installation. See "Setting the TERMINAL name" on page 506 for information on this.

  Note that when processing an AUTOINSTALL request for an LU6.2 single session terminal, the four byte terminal identifier returned by the user program

is used to name a CONNECTION. Therefore, the terminal identifer must conform to the naming standards for a CONNECTION (rather than a TERMINAL), as defined in the *CICS Resource Definition Guide*. The user program could identify an LU6.2 AUTOINSTALL request in one of the following ways:
  - Use a MODEL naming convention and examine the model name pointed to by fullword 3.
  - Test bytes 14 and 15 of the CINIT BIND, which is pointed to by fullword 5 for X'0602' (LU6.2).
- Set the return code to X'00'.

  On entry to the autoinstall control program, the return code always has a nonzero value. If you do not change this, the autoinstall request is rejected.

If you are using MTS, the z/OS Communications Server supplies the PRINTER and ALTPRINTER NETNAMEs, if specified.

The printers need not be installed at this stage; however, they must be installed before you use Print Key support. PRINTER and ALTPRINTER IDs override PRINTER and ALTPRINTER NETNAMEs.

Note that TERMID, PRINTER, and ALTPRINTER are the only attributes of the TERMINAL definition that can be set by the autoinstall control program; all other attributes must come from one of the following sources:

- The z/OS Communications Server LOGMODE entry (MODEENT)
- The autoinstall model TERMINAL definition
- The TYPETERM definition that it refers to
- The QUERY function
- Model names from z/OS Communications Server MDLTAB MDLENT and the NETNAMEs of the printers from z/OS Communications Server ASLTAB ASLENT (if you are using MTS).

Note:

1. The QUERY function overrides any extended attributes specified in the TYPETERM definition.
2. You cannot override information in the LOGMODE entry with the model TERMINAL and TYPETERM; they must match.

If your control program decides to reject the INSTALL request, it should return to CICS with a nonzero value in the return code.

Having completed processing, the control program must return to CICS by issuing an EXEC CICS RETURN command.

Selecting the autoinstall model

If you are using model terminal support to supply the model name (and the named model exists and is valid), CICS passes the model name to your autoinstall control program—you do not need to make any further selection.

As a general rule, all the models in the list passed to your program match the SNA data for the LU. That is, a viable TCT entry usually results from the use of any of the models. (The exception to this rule involves the z/OS Communications Server for SNA RUSIZE; if this value is incompatible, CICS issues an error message.) The default autoinstall control program merely picks the first model in the list. However, this model may not provide the attributes required in all cases. For

instance, you do not want a 3270 display device definition for a 3270 printer. Your control program must be able to select the model that provides the characteristics you require for this terminal—for example, security characteristics.

To save on storage, you should try to minimize the number of different models available to the control program, and the number of different TYPETERM definitions referenced by those models. If you are migrating your definitions from DFHTCT macros, look carefully at them and eliminate those that are unnecessarily different from others. Use the QUERY function for all devices that can support it. For bisynchronous devices, which do not support QUERY, one approach is to make the definition as straightforward as possible, with no special features.

If you need special models for special cases, you can use a simple mapping of, for example, NETNAME (generic or specific) to AUTINSTNAME. Your control program could go through a table of special case NETNAMEs, choosing the specified model for each. The default model would be used for any terminal not in the table. The list of models presented to the control program is in alphabetical order with one exception, which is described in the notes to "z/OS Communications Server MODEENT macro operands" on page 875.

Setting the TERMINAL name

The TERMINAL name must be unique, and one through four characters long. The TERMINAL name is the identifier CICS uses for the terminal. The NETNAME is the identifier z/OS Communications Server uses for the terminal

For a list of the acceptable characters, see the *CICS Resource Definition Guide*.

You might have transactions that depend on the terminals from which they are initiated, or to which they will be attached, having particular TERMINAL names. Some transactions are restricted to particular terminals and others behave in different ways, depending on the terminal. In some cases, the transaction may gather statistics about terminal use, using the TERMINAL name as a reference. The TERMINAL name may have meaning to those managing, using, or maintaining the network: it might, for instance, denote geographical location or departmental function.

The NETNAME is really more suitable for these purposes than the TERMINAL name, because it is eight characters in length. If you can use the NETNAME, the TERMINAL name can be randomly assigned by the autoinstall control program, and it does not matter if a terminal has a different TERMINAL name every time the user logs on. The control program is required, in this case, only to make the TERMINAL name unique within the system in which the terminal is to be autoinstalled. If the control program attempts to install a TCT entry for a TERMINAL name that already has a TCT entry, the installation is rejected, despite the fact that the terminal is eligible and a suitable model has been found. (By contrast, if the NETNAME already has a TCT entry, the terminal uses it and autoinstall can never be invoked.)

The default autoinstall control program creates the TERMINAL name from the last four nonblank characters of the NETNAME. This may not satisfy the requirement for uniqueness. One way of overcoming this problem is to use the EXEC CICS INQUIRE command from the control program, to determine whether the TERMINAL name is already in use. If it is, modify the last character and check again.

However, you may be in a situation where you must continue to use unique and predictable TERMINAL names for your terminals. Your control program must be able to assign the correct TERMINAL name to each terminal, every time the user logs on. Two possible approaches to this problem are:

- Devise another algorithm to generate predictable TERMINAL names from NETNAMEs
- Use a table or file to map TERMINAL names to NETNAMEs.

Devising an algorithm avoids the disadvantages of using a table or a file, but it might be difficult to ensure both uniqueness and predictability. If some of the information in the NETNAME is not needed by CICS, it can be omitted from the TERMINAL name. An algorithm is probably most appropriate in this situation.

Using a table has two disadvantages, each of which loses you some of the benefits of autoinstall: it takes up storage and it must be maintained. You could create a table in main temporary storage, so that it is placed in extended storage, above 16MB. You could use a **VSAM file** rather than a table, to avoid the storage problem. However, this might be slower, because of the I/O associated with a file. The table or file can contain information such as PRINTER and ALTPRINTER, and you can add information such as AUTINSTNAME for devices that need particular autoinstall models. (See “Selecting the autoinstall model” on page 505.)

Considerations for SNA dynamic alias names: If a CICS region is using dynamic LU aliases (that is, LUAPFX=*xx* is specified on the SNA APPL definition), selecting a unique TERMINAL name may be more complicated than otherwise. The following factors should be considered:

- The default programs use the last 4 characters of the NETNAME, which does not produce a repeatable TERMID for an LU that is assigned a dynamic LU alias. Consider using the network qualified name in the CINIT or BIND if it is important that the termid is repeatable for each logon.
- If you use the last 4 characters of the NETNAME, a dynamic LU alias produces a terminal id of 0001, 0002, and so on. Check that your RDO-defined terminals do not have such names, and if necessary change your autoinstall control program's logic. For example, you could use the last character of the NETID concatenated with the last 3 from the real network name.
- There is some new sample code in DFHZATDX and DFHZATDY that extracts the network qualified name, referenced as NQNAME, from the CINIT or BIND and uses the last character of the NETID and the last 3 characters of the real network name to provide an alternative TERMID.

  If this logic fails to create a termid for any reason it drops through to create the terminal id from the network name as usual. Note this code is enclosed within comments and is supplied only to illustrate how to extract the required information from the CINIT and BIND '0E' control vectors
- The sample code is also added in the form of comments to the C, COBOL, and PL/I versions of DFHZATDX. If you use these, note that:
  - The PL/I sample, DFHZPTDX, must be compiled with the PL/I compiler option LANGLVL(SPROG).
  - The COBOL sample, DFHZCTDX, must be compiled with compiler option TRUNC(OPT).

CICS action on return from the control program

On return from the autoinstall control program, CICS examines the return code, and uses the information to determine whether to complete the logon request.

If the return code is zero, and if the other required information supplied is satisfactory, CICS schedules the new resource for OPNDST in order to complete the logon request. If the installation process fails, then the control program is driven again, as though a DELETE had occurred. (See the section "The autoinstall control program at DELETE" on page 509 for details.) This is necessary to allow the program to free any allocations (for example, terminal identifiers) made on the assumption that this INSTALL request would succeed.

If the return code is not zero, then CICS rejects the connection request in the same way as it rejects an attempt by an unknown terminal to log on to CICS when autoinstall is not enabled.

For all autoinstall activity, messages are written to the transient data destination CADL. If an INSTALL fails, a message is sent to CADL, with a reason code. You can therefore check the output from CADL to find out why an autoinstall request failed.

If an autoinstall attempt fails for lack of an exact match, then details of the "best failure" match between a model and the BIND image are written to the CADL transient data destination.

The message takes the following form:

```
DFHZC6987 BEST FAILURE FOR NETNAME: nnnnnnnn,
          WAS MODEL_NAME: mmmmmmmm,
          CINIT BIND: cccccccc...,
          MODEL BIND: bbbbbbbb...,
          MISMATCH BITS: xxxxxxxx...
```

where

- 'nnnnnnnn' is the netname of the LU which failed to log on.
- 'mmmmmmmm' is the name of model that gave the best failure. (That is, the one that had the fewest bits different from the BIND image supplied by z/OS Communications Server.)
- 'cccccccc...' is the CINIT BIND image.
- 'bbbbbbbb...' is the model BIND image.
- 'xxxxxxxx...' is a string of hexadecimal digits, where 'xx' represents one byte, and each byte position represents the corresponding byte position in the BIND image. A bit set to '1' indicates a mismatch in that position between the BIND image from z/OS Communications Server and the BIND image associated with the model.

A suggested course of action is as follows:

1. Determine whether a model such as 'mmmmmmmm' is suitable. If there are several models that have identical BIND images, differing only in end-user options, then only the first such model is named in the above message. It will be up to your control program to make the choice, when the logmode table entry is corrected.
2. Identify the z/OS Communications Server logmode table entry that is being used.
3. Check that this logmode table entry is not successfully in use with other applications, so that to change it might cause this other use of it to fail.

4. Amend the logmode table entry by switching the bits corresponding to 1-bits in the mismatch string. That is, if the bit in the z/OS Communications Server BIND image corresponding to the bit position set to ‘1' in ‘xxxxxxxx...' above is ‘1', set it to ‘0'; if it is ‘0', set it to ‘1'.

The autoinstall control program at DELETE

To provide symmetry of control over the autoinstall process, the autoinstall control program is also invoked when:

- A session with a previously automatically-installed resource has been ended
- An autoinstall request was accepted by the user program, but the subsequent INSTALL process failed for some reason.

To make it easier for you to write your control program, these two events can be considered to be identical. (There is no difference in the environment that exists, or in the actions that might need to be performed.)

Invoking the control program at DELETE enables you to reverse the processes carried out at the INSTALL event. For example, if the control program at INSTALL incremented a count of the total number of automatically installed resources, then the control program at DELETE would decrement that count.

The communication area at DELETE for terminals

Input to the program is specified in a communication area, addressed by DFHEICAP.

The layout of the communication area is shown in Figure 48.

```
Fullword 1            Standard Header
  Byte  1             Function Code           (X'F1')
  Bytes 2 - 3         Component Code          Always 'ZC'
  Byte  4             Reserved                Always X'00'
Fullword 2            Terminal ID of terminal to be deleted
Fullword 3            NETNAME of terminal to be deleted
  Bytes 1-2           Delete netname length
  Bytes 3-4           Start of Delete netname ID
Next 15 bytes         Remainder of Delete netname ID
```

Figure 48. Autoinstall control program's communication area at DELETE. For terminals (including APPC single-session devices).

The parameter list contains the following information:

1. Standard Header. Byte 1 indicates the request type. For deletion of local terminals (including APPC single-session devices installed via CINIT requests) the value is X'F1'.

   Note: A value of X'F5' or X'F6' represents the deletion of a local APPC connection that was installed by a BIND request; see “The autoinstall control program at DELETE” on page 533. A value of X'FA' or X'FB' represents the deletion of a shipped terminal or connection; see “Autoinstall control program at DELETE” on page 550. A value of X'FC' represents the deletion of a client virtual terminal; see “The autoinstall control program at DELETE” on page 561.
2. The terminal identifier of the deleted resource, as shown in Table 27 on page 510.

Table 27. Table 1. Autoinstall control program's parameter list at DELETE

	1st byte	2nd byte	3rd byte	4th byte
First fullword	"F1"	"Z"	"C"	Reserved
Second fullword	ID of terminal to be deleted	ID of terminal to be deleted	ID of terminal to be deleted	ID of terminal to be deleted
Third fullword	Length of netname to be deleted	Length of netname to be deleted	First two bytes of netname	First two bytes of netname
Next 15 bytes	Remainder of netname	Remainder of netname	Remainder of netname	Remainder of netname

Note that the named resource has been deleted by the time the control program is invoked, and is not therefore found by any TC LOCATE type functions.

Naming, testing, and debugging your autoinstall control program

Naming of the autoinstall control program

CINIT starts a supplied, user-replaceable autoinstall control program, named DFHZATDX, for terminals and APPC single-session connections. If you write your own version of the control program, you can name it differently.

After the system has been loaded, to find the name of the autoinstall control program currently identified to CICS you can use the CICS Explorer **Regions** operations view, the **EXEC CICS INQUIRE AUTOINSTALL** command, or the **CEMT INQUIRE AUTOINSTALL** command.

The default name is DFHZATDX.

To change the current program, use one of these options:

- The **Attributes** tab on the CICS Explorer **Regions** operations view.
- The AIEXIT system initialization parameter. For guidance information about how to use AIEXIT, see AIEXIT system initialization parameter in Reference -> System definition.
- The **EXEC CICS SET AUTOINSTALL** command or the **CEMT SET AUTOINSTALL** command. For more information about these commands, see SET AUTOINSTALL in Reference > System programming and CEMT SET AUTOINSTALL in Reference > System definition.

Testing and debugging

To help you test the operation of your autoinstall control program, you can run the program as a normal terminal-related application.

To do so, define your program and initiate it from a terminal.

The parameter list passed to the program is described in “The autoinstall control program at INSTALL” on page 501. You can construct a dummy parameter list in your test program, upon which operations can be performed. Running your program on a terminal before you use it properly means that you can use the EDF transaction to help debug your program. You can also make the program interactive, sending and receiving data from the terminal.

If you find that CICS does not offer any autoinstall models to your program, you can create a test autoinstall program that forces the model name (AUTINSTNAME) you want. With a z/OS Communications Server buffer trace running, try to log the device on to CICS. If CICS does not attempt to send a BIND, check the following:

- Does the model TERMINAL refer to the correct TYPETERM? (Or alternatively, is the TYPETERM in question referred to by the correct TERMINAL definition?)
- Is the TERMINAL definition AUTINSTMODEL(YES or ONLY)?
- Have you installed the group containing the autoinstall models (TERMINAL and TYPETERM definitions)?

If CICS attempts to BIND, compare the device's CINIT RU to the CICS BIND, and make corrections accordingly.

It is very important that you ensure that the z/OS Communications Server LOGMODE table entries for your terminals are correct, rather than defining new autoinstall models to fit incorrectly coded entries. Bear in mind, while you are testing, that CICS autoinstall does not work if a LOGMODE entry is incorrectly coded.

Note that you cannot force device attributes by specifying them in the TYPETERM definition. For autoinstall, the attributes defined in the LOGMODE entry must match those defined in the model; otherwise the model will not be selected. You cannot define a terminal in one way to the z/OS Communications Server and in another way to CICS.

If your control program abends, CICS does not, by default, cause a transaction dump to be written. To cause a dump to be taken after an abend, your program must issue an EXEC CICS HANDLE ABEND command.

Sample autoinstall control programs for terminals

The CICS-supplied default autoinstall program is an assembler-language command-level program, named DFHZATDX. The source of the default program is provided in COBOL, PL/I, and C, as well as in assembler language.

The names of the supplied programs and their associated copy books, and the CICSTS52.CICS libraries in which they can be found, are summarized in Table 28. Note that the COBOL, PL/I, and C copy books each have an alias of DFHTCUDS.

Table 28. Autoinstall programs and copy books

Language	Member name	Alias	Library
Programs:			
Assembler	DFHZATDX	None	SDFHSAMP
COBOL	DFHZCTDX	None	SDFHSAMP
PL/I	DFHZPTDX	None	SDFHSAMP
C	DFHZDTDX	None	SDFHSAMP
Copy books:			
Assembler	DFHTCUDS	None	SDFHMAC
COBOL	DFHTCUD	DFHTCUDS	SDFHCOB
PL/I	DFHTCUDP	DFHTCUDS	SDFHPL1
C	DFHTCUD	DFHTCUDS	SDFHC370

The module generated from the assembler-language source program is part of the pregenerated library shipped in CICSTS52.CICS.SDFHLOAD. You can use it without modification, or you can customize it according to your own requirements. If you choose to alter the code in the sample program, take a copy of the sample and modify it. After modification, use the appropriate procedure to translate, assemble, and link-edit your module. Then put the load module into a user library that is concatenated before CICSTS52.CICS.SDFHLOAD in the DFHRPL statement. (This method applies to completely new modules as well as modified sample modules.) For more guidance information about the supplied procedures, see Using the CICS-supplied procedures to install programs in Deploying. Do not overwrite the sample with your customized module, because subsequent service may overwrite your module. You must install a new resource definition for a customized user program.

The default action of the sample program, on INSTALL, is to select the first model in the list, and derive the terminal identifier from the last four nonblank characters of the NETNAME, set the status byte, and return to CICS. If there are no models in the list, it returns with no action.

The default action, on DELETE, is to address the passed parameter list, and return to CICS with no action.

You can customize the sample program to carry out any processing that suits your installation. Examples of customization are given in "Customizing the sample program." Generally, your user program could:

- Count and limit the total number of logged-on terminals.
- Count and limit the number of automatically installed terminals.
- Keep utilization information about specific terminals.
- Map TERMINAL name and NETNAME.
- Map TNADDR (TCP/IP client address, IP port and, optionally, host name) of automatically installed terminals.
- Do general logging.
- Handle special cases (for example, always allow specific terminals or users to log on).
- Send messages to the operator.
- Exercise network-wide control over autoinstall. A network-wide, global autoinstall control program can reside on one CICS system. When an autoinstall request is received by a control program on a remote CICS system, this global control program can be invoked and data transferred from one control program to another.

Customizing the sample program

Before using any of the sample programs in a production environment, you must customize it to suit your installation.

Assembler language

Figure 49 on page 513, in assembler language, limits logon to netnames L77A and L77B. The model names used are known in advance. A logon request from any other terminal, or a request for a model which cannot be found, is rejected.

```
*    REGISTER CONVENTIONS =                                          *
*      R0  free                                                      *
*      R1  free                                                      *
*      R2  Base for Input parameters                                 *
*      R3  Base for code addressability                              *
*      R4  Base for model name list                                  *
*      R5  Base for output parameter list                            *
*      R6  Work register                                             *
*      R7  -----"-------                                             *
*      R8  -----"-------                                             *
*      R9  free                                                      *
*      R10 Internal subroutine linkage return                        *
*      R11 Base for EIB                                              *
*      R12 free                                                      *
*      R13 Base for dynamic storage                                  *
*      R14 free                                                      *
*      R15 free                                                      *
                                                                     *
* SELECT MODEL
*
         LH    R6,TABLEN                  Number of valid netnames
         LA    R7,TABLE                   Address the table
*
LOOP1    CLC   NETNAME(4),0(R7)           Is this netname in table?
         BE    VALIDT
*
         LA    R7,16(R7)                  Next table entry
         BCT   R6,LOOP1
*
*        Now we know its not a valid netname
*          return and the logon is rejected
*
         B     RETURN
*
```

Figure 49. Example of how to customize the DFHZATDX sample program (part 1)

```
*                                       R7 now points to model name
VALIDT   CLI   SELECTED_MODELNAME,C' '   MTS model name supplied?
         BNE   VALIDM1                   Yes
         LH    R6,MODELNAME_COUNT        Count of models
         LTR   R6,R6                     Were any presented?
         BZ    RETURN                    No
         LA    R8,MODELNAME              First model name
*
LOOP2    CLC   8(8,R7),0(R8)             Is this model name here?
         BE    VALIDM
*
         LA    R8,L'MODELNAME(R8)        Next model name
         BCT   R6,LOOP2
*
*        Now we know the required model name was not presented
*        to this exit by CICS, a return rejects the logon
*
         B     RETURN
*
*        At this point the model name was found in those presented
*        It is given to CICS and the new termid is
*        the netname
*
VALIDM   MVC   SELECTED_MODELNAME,0(R8)  R8 was left pointing at
*                                       model name
VALIDM1  DS    0H
         MVC   SELECTED_TERM_ID,NETNAME  Use netname for termid
*                                       (4 chars)
*
*
* SELECTIONS COMPLETE, RETURN
*
         MVI   SELECTED_RETURN_CODE,X'00'  Indicate all OK
         B     RETURN                    Exit program
*
*        Table of netnames allowed to log on and the model name
*        necessary for the logon to be successful
*
*        Format of table :
*            Bytes 1  to  8     Netname allowed to log on
*                  9  to  16    Model required for netname
*
         DS    0D
TABLE    DC    CL8'L77A',CL8'3270064'
         DC    CL8'L77B',CL8'3270065'
TABLEN   DC    Y((*-TABLE)/16)
*
```

Figure 50. Example of how to customize the DFHZATDX sample program (part 2)

COBOL

Figure 51 on page 515, in COBOL, redefines the NETNAME, so that the last four characters are used to select a more suitable model than that selected in the sample control program.

```
              .
*
* Redefine the netname so that the last 4 characters (of 7)
* can be used to select the autoinstall model to be used.
*
* The netnames to be supplied are known to be of the form:
*
*      HVMXNNN
*
* HVM is the prefix
* X   is the system name
* NNN is the address of the terminal
*
  01  NETNAME-BITS.
      02  FIRST-CHRS PIC X(3).
      02  NEXT-CHRS.
          03 NODE-LETTER PIC X(1).
          03 NODE-ADDRESS PIC X(3).
      02  LAST-CHR PIC X(1).
              .
              .
 PROCEDURE DIVISION.
              .
              .
*
* Select the autoinstall model to be used according to the
* node letter (see above). The models to be used are user
* defined.
*
* (It is assumed that the netname supplied in the commarea by CICS
* has been moved to NETNAME-BITS).
*
* If the node letter is C then use model AUTO2
* If the terminal netname is HVMC289 (a special case) then use
* model AUTO1.
* Otherwise (node letters A,B,D...) use model AUTO3.
*
     IF NODE-LETTER = 'C' THEN MOVE 'AUTO2' TO SELECTED-MODELNAME.
     IF NEXT-CHRS = 'C289' THEN MOVE 'AUTO1' TO SELECTED-MODELNAME.
     IF NODE-LETTER = 'A' THEN MOVE 'AUTO3' TO SELECTED-MODELNAME.
     IF NODE-LETTER = 'B' THEN MOVE 'AUTO3' TO SELECTED-MODELNAME.
     IF NODE-LETTER = 'D' THEN MOVE 'AUTO3' TO SELECTED-MODELNAME.
               .
               .
```

Figure 51. Example of how to customize the DFHZCTDX sample program

PL/I

Figure 52 on page 516, in PL/I, extracts information from the z/OS Communications Server CINIT RU, which carries the BIND image. Part of this information is the screen presentation services information, such as the default screen size and alternate screen size. The alternate screen size is used to determine the model of terminal that is requesting logon. The presented models are searched for a match, and if there is no match, the first model from those presented is used.

```
DCL 1 CINIT                    BASED(INSTALL_CINIT_PTR),
      2 CINIT_LENG              FIXED BIN(15),
      2 CINIT_RU                CHAR(256);
DCL   SAVE_CINIT               CHAR(256);
                              /* Temp save area for CINIT RU */
DCL 1 SCRNSZ                   BASED(ADDR(SAVE_CINIT)),
      2 SPARE                   CHAR(31),
                       /* Bypass first part of CINIT and reach */
                           /* into BIND image carried in CINIT */
      2 DHGT                    BIT(8),
                      /* Screen default height in BIND PS area */
      2 DWID                    BIT(8),
                       /* Screen default width in BIND PS area */
      2 AHGT                    BIT(8),
                     /* Screen alternate height in BIND PS area */
      2 AWID                    BIT(8);
                      /* Screen alternate width in BIND PS area */
DCL   NAME                     CHAR(2);
                         /* Used to work up a screen model type */
DCL   TERMID                   PIC'9999' INIT(1) STATIC;
                             /* Used to work up a unique termid */
DCL   ENQ                      CHAR(8) INIT('AUTOPRG');
                    /* Used to prevent multiple access to termid */
/* If model name supplied by MTS, bypass model name selection */
 IF SELECTED_MODELNAME ¬= '        '
   THEN GO TO MODEL_EXIT;
                    /* Clear the CINIT save area and move in the */
                          /* z/OS Communications Server CINIT RU.*/
        /* This is useful if you fail to recognize the model */
        /* of terminal; provide a dump and analyze this data */
 SAVE_CINIT = LOW(256);
 SUBSTR(SAVE_CINIT,1,CINIT_LEN) = SUBSTR(CINIT_RU,1,CINIT_LEN);
```

Figure 52. Example of how to customize the DFHZPTDX sample program (part 1)

```
        /* Now access the screen PS area in the portion of the BIND
           image presented in the CINIT RU */
        /* using the screen alternate height as a guide to the model
           of terminal attempting logon. If this cannot be determined
           then default to the first model in the table */
        SELECT (AHGT);               /* NOW GET SCRN ALTERNATE HEIGHT */
          WHEN (12)  NAME = 'M1';        /* MODEL 1 */
          WHEN (32)  NAME = 'M3';        /*       3 */
          WHEN (43)  NAME = 'M4';        /*       4 */
          WHEN (27)  NAME = 'M5';        /*       5 */
          OTHERWISE  NAME = 'M2';        /*       2 */
         END;
         /* Search the model entries for a matching entry.      */
         /* The criterion here is that a model definition should*/
         /* contain the chars M2 for a model 2, and so on.      */
         /* For example, L3270M2, L3270M5                       */
         /*     TERMM2,  TERMM5                                 */
         IF MODELNAME_COUNT = 0
         THEN GO TO EXIT;
         DO I = 1 TO MODELNAME_COUNT;
          IF INDEX(MODELNAME(I),NAME)
          THEN GO TO FOUND_MODEL;
         END;
NO_MODEL: /* Matching entry was not found, default to first model*/
         SELECTED_MODELNAME = MODELNAME(1);
         GO TO MODEL_EXIT;
FOUND_MODEL: /* Move the selected model name to the return area */
          SELECTED_MODELNAME = MODELNAME(I);
MODEL_EXIT: /* ENQ to stop multiple updates of counter.     */
            /* A simple counter is used to generate unique  */
            /* terminal identities, so concurrent access to */
            /* this counter is denied to ensure no two get  */
            /* the same identifier or update the counter.   */

  /* To use this method the program must be defined as resident.*/
          EXEC CICS ENQ RESOURCE(ENQ);
          SELECTED_TERMID = TERMID; /* Set SELECTED_TERMID to
                                       count value  */
          TERMID = TERMID + 1; /* Increase the count value by 1  */
          IF TERMID = 9999 THEN TERMID = 1;     /* Reset if too large*/
          EXEC CICS DEQ RESOURCE(ENQ);
NAME_EXIT:
          INSTALL_RETURN_CODE = LOW(1);
                               /* Set stat field to X'00' to allow
                                  logon to be processed */
          GO TO EXIT;
END INSTALL;
```

Figure 53. Example of how to customize the DFHZPTDX sample program (part 2)

Chapter 13. Writing a program to control autoinstall of consoles

You can write a program to control the automatic installation of MVS console devices, including TSO consoles.

For information about controlling the automatic installation of locally-attached SNA LUs, see Chapter 12, “Writing a program to control autoinstall of LUs,” on page 499.

Autoinstalling consoles - preliminary considerations

The reasons for using autoinstall for MVS consoles are the same as those that apply to the autoinstall for z/OS Communications Server for SNA devices: you don't have to define each device explicitly, and you save on storage (see “Autoinstalling terminals” on page 499).

How CICS autoinstalls consoles automatically

In addition to the normal autoinstall support, you can choose to allow CICS to autoinstall consoles without calling the autoinstall program.

If you specify autoinstall for consoles without calling the autoinstall program, CICS uses one of the following options:

- A model console definition with an AUTINSTNAME (model name) that matches the MVS console name
- The first suitable console model it finds in alphanumeric sequence

If the autoinstall control program is not called, CICS generates a 4-character terminal ID starting with the ¬ (logical not) symbol.

If you want CICS to install your consoles automatically you must perform the following actions:

- Specify AICONS=AUTO or use one of these options to autoinstall:
  - The **Attributes** tab on the CICS Explorer **Regions** operations view
  - The **CEMT SET AUTOINSTALL FULLAUTO** command
  - The **EXEC CICS SET AUTOINSTALL FULLAUTO** command
  - The **EXEC CICS SET AUTOINSTALL CONSOLES(1073)** command
- Create at least one model TERMINAL definition that references a TYPETERM definition specifying DEVICE(CONSOLE). You can use the IBM-supplied definition in group DFHTERMC if it suits your needs.
- Install the model TERMINAL and TYPETERM definition.

Using an autoinstall program

Use the default autoinstall control program or you can write your own program, by using the source code of the default program and the customization examples as a basis.

About this task

If you choose to have your autoinstall control program started for consoles, follow these steps:

Procedure

1. Use the default autoinstall control program for terminals (DFHZATDX or DFHZATDY), or write your own program, by using the source code of the default program and the customization examples as a basis. You can have only one active autoinstall control program to handle terminals, consoles, and APPC connections. Specify the name of the active program on the AIEXIT system initialization parameter. Your autoinstall program must be able to recognize the console INSTALL and DELETE parameter lists and return a model name, termid, and return code.
2. Enable the CICS AUTOINSTALL function for consoles, either by specifying AICONS=YES as a system initialization parameter, using the **Attributes** tab from the CICS Explorer **Regions** operations view, or by issuing a **SET AUTOINSTALL CONSOLES(PROGAUTO)** command.
3. Specify the AIEXIT system initialization parameter to define your autoinstall control program to CICS.

Results

If the AUTOINSTALL function has been enabled, CICS starts your autoinstall control program when the following conditions are satisfied:

- An autoinstall INSTALL request is being processed.
- An autoinstall request has previously been accepted by the autoinstall control program, but the subsequent INSTALL process has failed.
- The delay period since the console was last used has elapsed.

Autoinstall control program at INSTALL

If autoinstall is operative, you can specify that CICS is to invoke the autoinstall control program for MVS consoles, in addition to those devices listed in "The autoinstall control program at INSTALL" on page 501. To enable CICS to invoke the autoinstall control program for consoles, specify AICONS=YES as a system initialization parameter, or issue a **SET AUTOINSTALL CONSOLES(PROGAUTO)** command.

On each invocation of the autoinstall control program, CICS passes a parameter list to the control program by means of a communication area addressed by DFHEICAP. This information describes only the install function of console definitions.

The control program is invoked at INSTALL for a console when:

- CICS has received a MODIFY command from an MVS console whose console name is not defined to CICS.
- CICS has completed autoinstall processing to a point where it needs a terminal identifier and autoinstall model name, from the autoinstall control program, in order to process the CICS transaction passed on the MVS modify command.

The communication area at INSTALL for consoles

The layout of the communication area is shown in Figure 54 on page 521.

```
Fullword 1       Standard Header
  Byte  1        Function Code           (X'FD' for INSTALL)
  Bytes 2 - 3    Component Code          Always 'ZC'
  Byte  4        Reserved                Always X'00'
Fullword 2       Pointer to CONSOLENAME_FIELD
Fullword 3       Pointer to MODELNAME_LIST
Fullword 4       Pointer to SELECTED_PARMS
Fullword 5       Reserved
```

Figure 54. Autoinstall control program's communication area at INSTALL for consoles

The parameter list contains the following information:

1. A standard header. Byte 1 indicates the request type (this is hexadecimal character X'FD' for INSTALL), and bytes 2 to 3 contain the component code, which is always ZC for consoles. (Byte 4 is reserved.)
2. A pointer to a 2-byte length field, followed by the console name of the console which sent the message.
3. A pointer to an array of names of eligible autoinstall models. The array is preceded by a 2-byte field containing the number of 8-byte name elements in the array. If there are no elements in the array, the number field is set to zero.
4. A pointer to the area of storage that you use to return information to CICS.

CICS passes a list of eligible autoinstall models in the area addressed by fullword 3 of the parameter list. From this list, the control program must select a model that is suitable for the console device, and move the model name to the first 8 bytes of the area addressed by fullword 4 of the parameter list. Before returning to CICS, the control program must supply a CICS 4-character terminal ID for the console being logged on, and set the return code field to X'00' if the autoinstall request is to be allowed. Your program can also set the delay period that is to follow the last use of a console before it is automatically deleted by CICS. On entry to your autoinstall control program, this value is set to a default value of 60 minutes. Override this by storing your own delay period, in minutes, as a fullword binary value. Setting this field to zero (0) means that CICS never deletes the console.

Figure 55 on page 522 shows all of these fields in their required order.

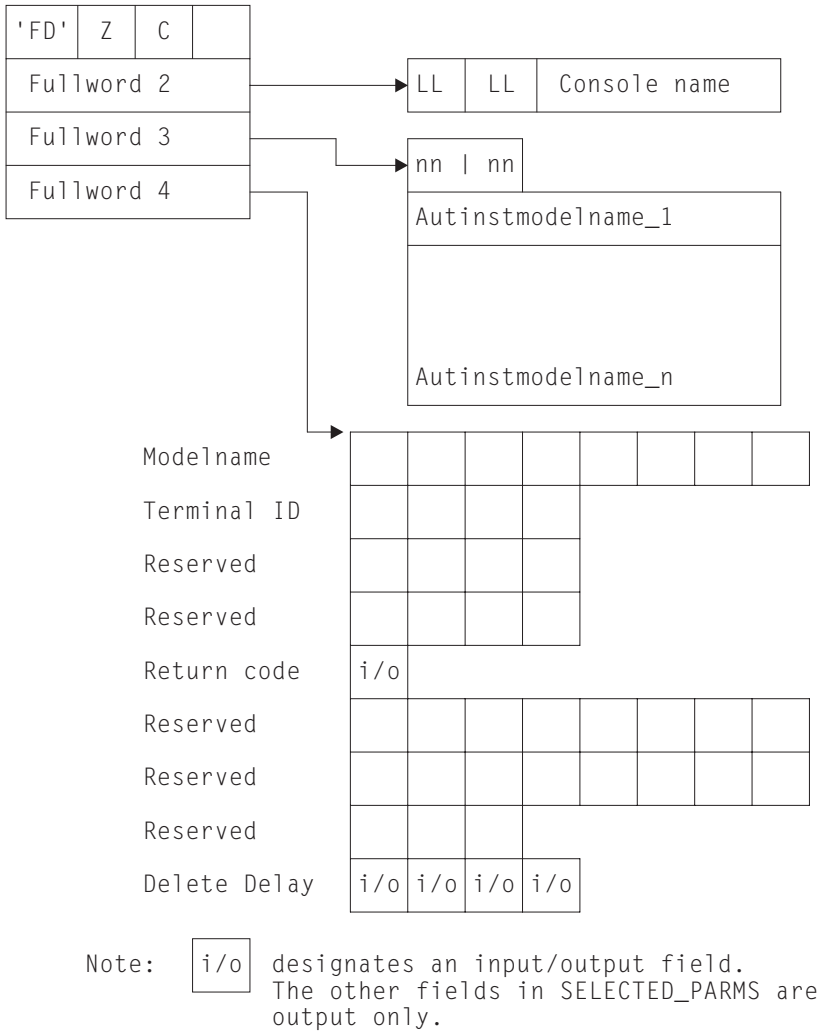


Figure 55. Autoinstall control program's parameter list at INSTALL

How CICS builds the list of autoinstall models

CICS builds the list of autoinstall models by selecting from its complete list of terminal models those models that define console devices.

The complete list of autoinstall models available to CICS at any time comprises all the definitions with AUTINSTMODEL(YES) and AUTINSTMODEL(ONLY) that are installed, and which reference a TYPETERM definition that specifies DEVICE(CONSOLE).

If CICS cannot find a model for consoles, it issues message DFHZC6902. If the return code in the area addressed by fullword 4 of the parameter list is nonzero, CICS issues error message DFHZC6987.

You can obtain a list of autoinstall model definitions by using the CICS Explorer **Regions** operations view, CEMT, or **EXEC CICS INQUIRE AUTINSTMODEL** commands.

Returning information to CICS

At the INSTALL event, the autoinstall control program is responsible for allowing or denying the installation of a new console resource in the CICS region. This decision can be based on a number of installation-dependent factors, such as security, or the total number of connected terminals. CICS takes no part in any such checking. You decide whether any such checking takes place, and how it is done.

About this task

If you want an INSTALL request to proceed, perform these steps in your autoinstall control program:

- Return an autoinstall model name in the first 8 bytes of the area addressed by fullword 4 of the parameter list.
- Supply a CICS terminal name (TERMID) in the next four bytes of the return area. DFHZATDX and DFHZATDY take the last four non-blank characters of the console name (addressed by fullword 2 of the parameter list) as the terminal name. If this does not meet with your installation's naming conventions, code your own autoinstall program.
- Set the return code to X'00'. On entry to the autoinstall control program, the return code always has a nonzero value. If you do not change this, the autoinstall request is rejected.
- Set the delete delay period, or leave it set to the default value of 60 minutes.

Note that these are the only attributes of the TERMINAL definition that can be set by the autoinstall control program; all other attributes must come from one of the following sources:

- The MVS console interface block (CIB)
- The autoinstall model TERMINAL definition
- The TYPETERM definition to which it refers.

If your control program decides to reject the INSTALL request, it should return to CICS with a non-zero value in the return code. Having completed processing, the control program must return to CICS by issuing an EXEC CICS RETURN command.

Selecting the autoinstall model

All the models in the list passed to your program are for consoles. That is to say, a viable TCT entry usually results from the use of any one of them. The default autoinstall control program picks the first model in the list. However, this model may not provide the attributes required in all cases. Your control program must be able to select the model that provides the characteristics you require for the console—for example, one that has the required security characteristics.

Setting the TERMINAL value

The TERMINAL value must be unique, and must be 1 - 4 characters long. The TERMINAL value is the name or identifier that CICS uses for the console. The CONSNAME value is the identifier MVS uses for the console.

If the control program attempts to install a TCT entry for a TERMINAL value that already has a TCT entry, the installation is rejected, even if the terminal is eligible and a suitable model has been found. However, if a MODIFY command is received

from an MVS console for which CICS already has an entry in the TCT with a matching CONSNAME value, CICS uses that entry and does not start your autoinstall control program.

The default autoinstall control program creates the TERMINAL value from the last four non-blank characters of the CONSNAME value, which means that the terminal name might not be unique. One way of overcoming this problem is to use the CICS Explorer **Terminals** operations view or the **EXEC CICS INQUIRE** command from the control program, to determine whether the TERMINAL value is already in use. If it is, modify the last character and check again.

CICS action on return from the control program

When CICS receives control back from the autoinstall control program, it examines the return code field:

- If the return code is zero, and the other required information supplied is satisfactory, CICS schedules the transaction specified on the MODIFY command to complete the request. If the installation process fails, the autoinstall control program is driven again, as though a DELETE function is being processed.
- If the return code is not zero, CICS rejects the connection request in the same way that it rejects an attempt by an unknown console to send a modify request to CICS when autoinstall is not enabled.

For all autoinstall activity, messages are written to the transient data destination CADL. If an INSTALL fails, a message is sent to CADL, with a reason code. You can check the output from CADL to find out why an autoinstall request failed.

The autoinstall control program at DELETE

The autoinstall control program can be started when a console autoinstall request fails or when the delete delay period has expired.

To provide symmetry of control over the autoinstall process, the autoinstall control program is also started whenthe following situations occur:

- A console autoinstall request was accepted by the user program, but the INSTALL process failed.
- The delete delay period has passed since the console was last used and CICS is running with AICONS=YES in effect. You can query this status of autoinstall for consoles by using the CICS Explorer **Regions** operations view or by issuing a **CEMT INQUIRE AUTOINSTALL** command. If AICONS=YES is specified, the value of the CONSOLES option is PROGAUTO.

Input to the program is through a communication area, addressed by DFHEICAP. The layout of the communication area is shown in the following figure:

```
Fullword 1         Standard Header
  Byte  1          Function Code           (X'FE')
  Bytes 2 - 3      Component Code          Always 'ZC'
  Byte  4          Reserved                Always X'00'
Fullword 2         Terminal ID of console being deleted
Fullword 3         Consolename of console being deleted
  Bytes 1-2        Deleted consolename length
  Bytes 3-4        Start of deleted consolename ID
Next 6 bytes       Remainder of deleted consolename ID
```

Figure 56. Autoinstall control program communication area at DELETE for consoles

The parameter list contains the following information:

1. A standard header, where byte 1 indicates the request type (the hexadecimal character X'FE' represents DELETE), and bytes 2 - 3 contain the component code, which is always ZC for consoles.
2. The second fullword contains the terminal ID of the console that is being deleted.
3. The third fullword contains, in the first 2 bytes, the length of the deleted console name and, in the last 2 bytes, the first and second characters of the console name.
4. The last 6 bytes of the communications area contain the remainder of the console name (third to eighth characters).

Sample autoinstall control programs for consoles

CICS supplies a default autoinstall control program, written in each of the supported programming languages, all of which contain the necessary support for handling consoles.

For details of the programs, see “Sample autoinstall control programs for terminals” on page 511.

Chapter 14. Writing a program to control autoinstall of APPC connections

You can write a program to control the automatic installation of local APPC connections.

For information about controlling the automatic installation of local SNA LUs, see Chapter 12, “Writing a program to control autoinstall of LUs,” on page 499. For information about controlling the installation of shipped terminals and connections, see Chapter 16, “Writing a program to control autoinstall of shipped terminals,” on page 545. For information about controlling the installation of virtual LUs used by CICS clients, see Chapter 17, “Writing a program to control autoinstall of virtual terminals,” on page 553.

Autoinstalling APPC connections - preliminary considerations

In considering the autoinstall of local APPC connections, you must distinguish between the following:

1. Local APPC single-session connections initiated by CINIT requests
2. Local APPC parallel- and single-session connections initiated by incoming bind requests. (By “incoming” we mean that the request is initiated by the partner system.)

Local APPC single-session connections initiated by CINIT

Autoinstall of local APPC single-session connections that are initiated by CINIT requests works in the same way as autoinstall for terminals. You must provide a TERMINAL—TYPETERM model pair, and a customized version of one of the supplied autoinstall control programs, DFHZATDX or DFHZATDY.

See Chapter 12, “Writing a program to control autoinstall of LUs,” on page 499.

Local APPC parallel-session and single-session connections initiated by BIND

If autoinstall is enabled, and an incoming APPC BIND request is received for an APPC service manager (SNASVCMG) session (or for the only session of a single-session connection), and there is no matching CICS CONNECTION definition, a new connection is created and installed automatically.

Like autoinstall for other resources, autoinstall for APPC connections requires model definitions. However, unlike the model definitions used to autoinstall terminals, those used to autoinstall APPC links do not need to be defined explicitly as models. Instead, CICS can use any previously-installed connection definition as a “template” for a new definition. In order for autoinstall to work, you must have a template for each kind of connection you want to be autoinstalled.

Autoinstall templates for APPC connections

The purpose of a template is to provide CICS with a definition that can be used for all connections with the same properties. You customize the supplied autoinstall

control program, DFHZATDY, to select an appropriate template for each new connection, depending on the information it receives from the z/OS Communications Server for SNA.

A template consists of a CONNECTION definition and its associated SESSIONS definitions. You should have a definition installed for each different set of session properties you are going to need.

Any installed connection definition can be used as a template, but for performance reasons, your template should be an installed connection definition that you do not use. The definition is locked while CICS is copying it, and if you have a very large number of sessions autoinstalling, the delay may be noticeable.

Benefits of autoinstall

Autoinstall support is likely to be beneficial if you have large numbers of APPC parallel session devices with identical characteristics.

For example, if you had 1000 personal computers (PC)s, all with the same characteristics, you would set up one template to autoinstall all of them. If 500 of your PCs had one set of characteristics, and 500 had another set, you would set up two templates to autoinstall them.

Restart of any kind should be noticeably faster, especially when large numbers of terminals are involved.

Savings can also be made on systems management overheads, and on storage, as autoinstalled resources do not occupy space before they are used.

Requirements for autoinstall

Autoinstall of APPC connections works with any supported release of ACF/SNA.

You can have only one active autoinstall control program for terminals and connections. You must specify the name of the active program on the AIEXIT system initialization parameter. As well as providing function to autoinstall APPC connections initiated by BIND requests, the sample program, DFHZATDY, provides the same function for terminal autoinstall as the default control program, DFHZATDX, described in Chapter 12, “Writing a program to control autoinstall of LUs,” on page 499. Thus, you can use a customized version of DFHZATDY to autoinstall both terminals and APPC connections.

Note: Both DFHZATDX and DFHZATDY provide function to install shipped terminals and connections, and Client virtual terminals.

You may find the supplied version of DFHZATDY adequate for your purposes. If not, you can write a customized version of the supplied program, or create your own program to provide enhanced function.

The autoinstall control program for APPC connections

The purpose of the autoinstall control program is to provide CICS with any extra information it needs to complete an autoinstall request. For APPC connections, the control program selects the template to be used, and provides a name for the new connection.

If autoinstall is enabled, when CICS receives an APPC BIND request for an SNASVCMG session (or for the only session of a single-session connection), if there is no matching CONNECTION definition, CICS passes the partner's z/OS Communications Server NETNAME to the autoinstall control program. The control program uses information from the BIND, which is passed in the communications area, to select the most appropriate template on which to base a new connection.

The control program needs to know the NETNAME or SYSID of all the templates, in order to return the name of the most suitable one. If it attempts to use an unsuitable template, message DFHZC6922 is issued, explaining why the template is unusable.

If the template is usable, CICS makes a copy of the definitions within it and attempts to install the new CONNECTION definition. If the installation is not successful, message DFHZC6903 is issued.

Recovery and restart

Autoinstalled connections are not cataloged by CICS, so they are not recovered at an emergency restart or a warm restart.

Autoinstall control program at INSTALL

The autoinstall control program is invoked at INSTALL for:

- Local SNA LUs
- MVS consoles
- Local APPC single-session connections initiated by a CINIT
- Local APPC parallel-session connections initiated by a BIND
- Local APPC single-session connections initiated by a BIND
- Shipped terminals and connections
- Client virtual terminals.

On each invocation, CICS passes a parameter list to the control program by means of a communication area addressed by DFHEICAP. The parameter list passed at INSTALL of local terminals and APPC single-session connections initiated by CINIT is described in “The communication area at INSTALL for terminals” on page 502. The parameter list passed at INSTALL of MVS consoles is described in “Autoinstall control program at INSTALL” on page 520. The parameter list passed at INSTALL of shipped terminals and connections is described in “The communications area at INSTALL for shipped terminals” on page 547. The parameter list passed at INSTALL of Client virtual terminals is described in “The communications area at INSTALL for Client virtual terminals” on page 557. This section describes only INSTALL of local APPC connections initiated by BIND requests.

The communication area at INSTALL for APPC connections

The communications area is mapped by the DSECT for the assembler version of DFHZATDY, which is supplied in CICSTS52.CICS.SDFHMAC.

```
*-------------------------------------------------------------------*
* APPC Install parameter list - Functions 2, 3, and 4               *
*-------------------------------------------------------------------*
INSTALL_APPC_COMMAREA       DSECT       Install Parameter List
*
INSTALL_APPC_STANDARD       DS  F       Standard field
                            ORG INSTALL_APPC_STANDARD
INSTALL_APPC_EXIT_FUNCTION  DS  XL1     Install request type
INSTALL_APPC_PS_CINIT       EQU X'F2'   Install PS via CINIT
INSTALL_APPC_PS_BIND        EQU X'F3'   Install PS via BIND
INSTALL_APPC_SS_BIND        EQU X'F4'   Install SS via BIND
INSTALL_APPC_EXIT_COMPONENT DS  CL2     Component ID 'ZC'
                            DS  XL1     Reserved
*
                            ORG ,
INSTALL_APPC_NETNAME_PTR    DS  A       -> NETNAME          Input
INSTALL_APPC_CINIT_PTR      DS  0A      -> CINIT_RU         Input
INSTALL_APPC_BIND_PTR       DS  A       -> BIND             Input
INSTALL_APPC_SELECTED_PTR   DS  A       -> Return fields    Output
INSTALL_APPC_SYNCLEVEL_PTR  DS  A       -> Sync level       Input
*
INSTALL_APPC_TEMPLATE_NETNAME_PTR DS A  -> Template NETNAME I/O
INSTALL_APPC_TEMPLATE_SYSID_PTR DS A    -> Template SYSID   Output
INSTALL_APPC_SYSID_PTR      DS A        -> New SYSID        Output
INSTALL_APPC_NETNAME2_PTR   DS A        -> Generic or       Input
*                                          member NETNAME
INSTALL_APPC_NETID_PTR      DS A        -> Network ID of    Input
*                                          incoming bind
INSTALL_APPC_TYPE_PTR       DS A        -> Generic          Input
                                           resource type

*
TEMPLATE_NETNAME            DS CL8      Put netname of template here
TEMPLATE_SYSID              DS CL4      Put sysid of template here
SYSID                       DS CL4      Put name of new connection here
SYNCLEVEL                   DS XL2      Synclevel of new connection
APPC_NETID                  DS CL8      NETID of incoming bind
APPC_GR_TYPE                DS CL1      G = NETNAME is generic resource name
                                        M = NETNAME is member name
                                        blank = This CICS is not a generic
                                            resource or the partner is not a
                                            generic resource.
APPC_NETNAME2_FIELD         DSECT
APPC_NETNAME2_LENGTH        DS XL2      Length of NETNAME
APPC_NETNAME2               DS 0X       Generic or member NETNAME
```

Figure 57. Autoinstall control program's communications area at INSTALL. For APPC connections initiated by BIND requests.

INSTALL_APPC_STANDARD header
A fullword input field comprising the following information:

INSTALL_APPC_EXIT_FUNCTION
A 1-byte field that defines the install request type. The equated values are:

INSTALL_APPC_PS_CINIT
X'F2' represents an install request for an APPC parallel-session connection from a secondary node via a CINIT request.

Note: These requests cannot be received by CICS Transaction Server for z/OS, Version 5 Release 2.

INSTALL_APPC_PS_BIND
X'F3' represents an install request for an APPC parallel-session connection via a BIND.

INSTALL_APPC_SS_BIND
X'F4' represents an install request for an APPC single-session connection via a BIND.

Note: The values X'F0' and X'F1' represent, respectively, install and delete requests for terminals (including APPC single-session devices). See Chapter 12, “Writing a program to control autoinstall of LUs,” on page 499.

INSTALL_APPC_EXIT_COMPONENT
A 2-byte component code, which is set to ‘ZC'.

INSTALL_APPC_NETNAME_PTR
A fullword pointer to a 2-byte length field, followed by the NETNAME to be installed (input field).

For connections to CICS TORs where the partner is a generic resource, NETNAME can be the partner's generic resource name, or its member name, depending on the setting of APPC_GR_TYPE. (For introductory information about generic resources, see the *CICS Intercommunication Guide*.)

INSTALL_APPC_CINIT_PTR
A fullword pointer to an input field containing the incoming CINIT, if the incoming session is a secondary.

Note: Not applicable to CICS Transaction Server for z/OS, Version 5 Release 2.

INSTALL_APPC_BIND_PTR
A fullword pointer to a 2-byte length field, followed by an input field containing the incoming BIND.

INSTALL_APPC_SELECTED_PTR
A fullword pointer to the return fields. These are in the same format as those for autoinstall of terminals.

Note that for APPC autoinstall (functions X'F3' and X'F4') only the return code is used. You return other information for APPC in other fields defined in the communications area.

INSTALL_APPC_SYNCLEVEL_PTR
A fullword pointer to a 2-byte input field specifying the syncpoint level for the connection, which is extracted from the BIND. The possible values are:

X'0000'
Synclevel 0

X'0001'
Synclevel 1

X'0002'
Synclevel 2.

INSTALL_APPC_TEMPLATE_NETNAME_PTR
A fullword pointer to an 8-byte input/output area (TEMPLATE_NETNAME). On invocation, TEMPLATE_NETNAME normally contains blanks. However, if both the partner and the local CICS are registered as generic resources, it contains the NETNAME of the generic resource name connection, if one is present. (Generic resource name connections are described in the *CICS Intercommunication Guide*.)

Your control program can use the TEMPLATE_NETNAME field to specify the NETNAME of the template. For connections between generic resources, your

program can accept the suggested template passed by CICS, or specify a different one—either in this field or by overwriting the suggested template with blanks and putting a value in the TEMPLATE_SYSID field.

If the specified name is less than 8 bytes, it must be padded with trailing blanks. If, as an alternative to specifying the NETNAME of the template, your program specifies its CONNECTION name in TEMPLATE_SYSID, it should fill TEMPLATE_NETNAME with blanks.

INSTALL_APPC_TEMPLATE_SYSID_PTR
A fullword pointer to a 4-byte output area (TEMPLATE_SYSID) that your control program can use to specify the SYSID (connection name) of the template. If the name is less than 4 bytes, it must be padded with trailing blanks. If, as an alternative to specifying the SYSID of the template, your program specifies its NETNAME in TEMPLATE_NETNAME, it should fill TEMPLATE_SYSID with zeros.

INSTALL_APPC_SYSID_PTR
A fullword pointer to a 4-byte output area in which your program must put the SYSID for the new autoinstalled connection. The name you supply must be unique. You can use the same or similar logic to create it that you use for creating a terminal ID. If the name is less than 4 bytes, it must be padded with trailing blanks.

If you are using recoverable resources, the SYSID chosen for a connection after a restart must be the same as that chosen in the previous CICS run.

INSTALL_APPC_NETNAME2_PTR
A fullword pointer to a 2-byte length field, followed by an 8-byte input field (APPC_NETNAME2).

If both the partner and the local CICS are generic resources, APPC_NETNAME2 is the partner's generic resource name or member name, depending on the setting of APPC_GR_TYPE.

If the partner is not a generic resource, APPC_NETNAME2 contains the same value as NETNAME.

If the local CICS is not a generic resource, the value of APPC_NETNAME2 is meaningless.

INSTALL_APPC_NETID_PTR
A fullword pointer to an 8-byte input field containing the Network ID of the partner. This field is set whenever the local CICS is registered as a generic resource. At all other times it has a value of 0.

INSTALL_APPC_GR_TYPE_PTR
A fullword pointer to a 1-byte input field indicating whether this is a connection between generic resources and, if so, whether the NETNAME passed on the BIND is the partner's generic resource name or its member name. The equated values are:

G NETNAME is the partner's generic resource name and APPC_NETNAME2 is its member name (applid).

M NETNAME is the partner's member name (applid) and APPC_NETNAME2 is its generic resource name.

Blank This CICS is not registered as a generic resource or the partner is not registered.

The autoinstall control program at DELETE

To provide symmetry of control over the autoinstall process, the autoinstall control program is also invoked when an autoinstalled APPC connection is deleted.

Invoking the control program at DELETE enables you to reverse the processes carried out at the INSTALL event. For example, if the control program at INSTALL incremented a count of the total number of automatically installed resources, then the control program at DELETE would decrement that count.

Input to the program is by a communication area, addressed by DFHEICAP. The layout of the communication area is shown in Figure 58.

```
Fullword 1            Standard Header
  Byte  1             Function Code        (X'F5' or X'F6')
  Bytes 2 - 3         Component Code       Always "ZC"
  Byte  4             Reserved             Always X'00'
Fullword 2            SYSID of deleted connection
Fullword 3            NETNAME of deleted connection
  Bytes 1-2           NETNAME length
  Bytes 3-10          NETNAME
```

Figure 58. Autoinstall control program's communication area at DELETE. For APPC connections initiated by BIND requests.

The Function Code byte (byte 1 of fullword 1) indicates why the user program has been invoked:

X'F5' After deletion of a parallel-session APPC connection that was initiated by a BIND.

X'F6' After deletion of a single-session APPC connection that was initiated by a BIND.

Note: The value X'F1' represents the deletion of a local terminal, or an APPC single-session device that was autoinstalled via a CINIT request. For more information, see “The autoinstall control program at DELETE” on page 509. The value X'FA' or X'FB' represents the deletion of a shipped terminal or connection. For more information, see “Autoinstall control program at DELETE” on page 550. The value X'FC' represents the deletion of a Client virtual terminal. For more information, see “The autoinstall control program at DELETE” on page 561.

When autoinstalled APPC connections are deleted

Any autoinstalled APPC connection entry is deleted if the connection is discarded. Connection entries can also be deleted when the terminal or system logs off, or is disconnected from CICS.

This type of implicit deletion occurs for the following types of APPC autoinstalled connection:

- Single-session connections installed by a CINIT.

  These connections are deleted when the terminal user logs off, after the expiry of the AILDELAY system initialization value (see AILDELAY system initialization parameter in Reference -> System definition).
- Synclevel 1 connections installed by a BIND.

  Most synclevel 1-only APPC connections that are autoinstalled through a BIND request are implicitly deleted at the following times:

- When the connection is released
- If SNA abends
- When CICS closes the SNA ACB
- After the expiry of the AIRDELAY interval following a warm or emergency start (if the value of the AIRDELAY system initialization parameter is greater than zero). See AIRDELAY system initialization parameter in Reference -> System definition.

However, this does not apply to limited resource connections that are installed on a CICS generic resource member. See Synclevel 2 connections installed by a BIND.

- Synclevel 2 connections installed by a BIND.

  Synclevel 2-capable APPC connections that are installed through a BIND request are implicitly deleted *only* if they are installed on a CICS generic resource member, and an affinity is ended. Otherwise, they are never implicitly deleted.

  The same applies to synclevel 1-only, limited resource connections that are installed on a CICS generic resource member.

Sample autoinstall control program for APPC connections

The sample control program for autoinstall of APPC connections is DFHZATDY. The source code, in assembler-language only, is in library CICSTS52.CICS.SDFHSAMP.

As well as providing function to autoinstall APPC connections initiated by BIND requests, DFHZATDY provides the same function for terminal autoinstall as the DFHZATDX program described in Chapter 12, “Writing a program to control autoinstall of LUs,” on page 499. You can use a customized version of DFHZATDY to autoinstall both terminals and APPC connections.

Default actions of the sample program

The role of DFHZATDY in installing APPC connections is to choose the template to be used (by supplying its NETNAME or SYSID), and to supply the name (SYSID) of the new connection.

The actions taken by the supplied version of the program are to:

1. Examine the request type passed in the INSTALL_APPC_EXIT_FUNCTION field:

   X'F0' An incoming CINIT for a terminal or APPC single-session device. Proceed as for DFHZATDX. See Chapter 12, “Writing a program to control autoinstall of LUs,” on page 499.

   X'F1' A delete request for a terminal or APPC single-session device. Proceed as for DFHZATDX. See Chapter 12, “Writing a program to control autoinstall of LUs,” on page 499.

   INSTALL_APPC_PS_CINIT (X'F2')
   An incoming CINIT for an APPC parallel-session connection. Specify a template by setting the field pointed to by INSTALL_APPC_TEMPLATE_SYSID to 'CCPS'.

   Note: This type of request cannot be received by CICS Transaction Server for z/OS, Version 5 Release 2.

INSTALL_APPC_PS_BIND (X'F3')
An incoming BIND for an APPC parallel-session connection. Specify a template. This is done in one of two ways:

- For connections between two generic resources, by accepting the suggested template (the generic resource name connection) whose NETNAME is passed in TEMPLATE_NETNAME. If there is no generic resource name connection, set TEMPLATE_SYSID to 'CBPS'.
- In all other cases, by setting TEMPLATE_SYSID to 'CBPS'.

INSTALL_APPC_SS_BIND (X'F4')
An incoming BIND for an APPC single-session connection. Specify a template by setting the field pointed to by INSTALL_APPC_TEMPLATE_SYSID to 'CBSS'.

X'F5' A delete request for an APPC parallel-session connection installed by a BIND. Establish addressability to the COMMAREA and return.

X'F6' A delete request for an APPC single-session connection installed by a BIND. Establish addressability to the COMMAREA and return.

2. Specify a name for the new connection by copying the last 4 non-blank characters of the input NETNAME pointed to by INSTALL_APPPC_NETNAME_PTR to the field pointed to by INSTALL_APPC_SYSID_PTR.
3. Indicate that a selection has been made by setting the return code to RETURN_OK.

Resource definitions

CICS supplies a resource definition group called DFHAI62, which defines DFHZATDY, and contains CONNECTION definitions for CCPS, CBPS, and CBSS.

If you want to use the supplied version of DFHZATDY, you should append DFHAI62 to your CICS startup grouplist. However, if you customize DFHZATDY you will probably need to create your own definitions.

DFHZATDY is defined as follows in DFHAI62:

```
DEFINE PROGRAM(DFHZATDY)
DESCRIPTION(Assembler definition for sessions autoinstall control program)
GROUP(DFHAI62)
LANGUAGE(ASSEMBLER) RELOAD(NO)          RESIDENT(NO)
USAGE(NORMAL)       STATUS(ENABLED)     CEDF(NO)
DATALOCATION(ANY)   EXECKEY(CICS)       EXECUTIONSET(FULLAPI)
```

Chapter 15. Writing a program to control autoinstall of IPIC connections

You can write a program to control the automatic installation of IPIC connections.

For introductory information about the different types of CICS intercommunication links, including IPIC, see Intercommunication methods in Getting started.

Autoinstalling IPIC connections; preliminary considerations

The IPCONN autoinstall user program is similar to the APPC autoinstall user program. Like the APPC autoinstall user program, the IPCONN autoinstall user program can choose an installed connection to use as a template for the new connection. The main differences are that the template is an IPCONN definition rather than a CONNECTION definition, and that the use of the template is optional.

If IPCONN autoinstall is active, CICS installs the new IPIC connection using this information:

- The information in the connect flow
- The IPCONN template, optionally selected by the IPCONN autoinstall user program
- Values returned by the user program in its communications area
- CICS-supplied values

Autoinstall templates for IPCONN resources

Unlike autoinstall for other resources, autoinstall for IPCONN resources does not require model definitions, although they are recommended. However, unlike the model definitions used to autoinstall terminals, those used to autoinstall IPCONN resources do not have to be defined explicitly as models. Instead, CICS can use any previously installed IPCONN definition as a template for a new definition.

The purpose of a template is to provide CICS with a definition that can be used for all connections with the same properties. You customize the supplied autoinstall user program to select an appropriate template for each new connection, depending on the information it receives from CICS.

You can use any installed IPCONN definition as a template but, for performance reasons, use an installed definition that you do not use as an appropriate template. The definition is locked while CICS is copying it, and, if you have a large number of IPCONNs being autoinstalled at one time, the delay might be noticeable.

Requirements for autoinstall

IPCONN autoinstall is active if these conditions are met:

1. The receiving region must have installed at least one TCPIPSERVICE that specifies PROTOCOL(IPIC).
2. The name of the IPCONN autoinstall user program must be specified on the URM option of the installed TCPIPSERVICE definition.

Note: This requirement differs from autoinstall of APPC connections, where the name of the autoinstall user program is specified on the AIEXIT system initialization parameter. IPCONN autoinstall has no equivalent system initialization parameter. Instead, the name of the autoinstall user program is specified on the TCPIPSERVICE definition.

As with APPC, putting the template IPCONNs out-of-service disables the autoinstall function.

When the user program is called

The user program is called when both the following conditions are met:

1. A TCPIPSERVICE resource that is defined with PROTOCOL(IPIC) receives either a connect flow containing a NETWORKID and APPLID combination for which there is no installed IPCONN definition, or a connect flow with a null APPLID. If HOST(ANY) is specified in the TCPIPSERVICE definition of the receiving CICS system instead of a specific IPv6 address, the IPCONN uses the default IPv4 address so that communication is guaranteed.
2. The URM attribute of the receiving TCPIPSERVICE resource specifies the name of an autoinstall user program. If the URM attribute contains `NO`, the autoinstall request is rejected.

The autoinstall user program for IPCONN resources

The purpose of the autoinstall user program is to provide CICS with any extra information it needs to complete an autoinstall request. For IPIC connections, the user program provides a name for the new connection. Optionally, it can select an in-service IPCONN definition to use as a template, and modify the values of the APPLID, HOST, and PORT attributes of the new connection from those supplied on the connect flow.

The RECEIVECOUNT attribute on the autoinstalled IPCONN resource is set to the value requested on the connect flow from the client, or the minimum of this value and the RECEIVECOUNT value from the template, if a template is specified. All other attributes of the new IPCONN definition are taken from the template, or from CICS-supplied values if no template is specified, and cannot be modified by the user program.

If the selected template is usable, CICS makes a copy of the definition in it and attempts to install the new IPCONN definition. If the installation is not successful, a message is issued.

The default autoinstall user program, DFHISAIP, is an assembler-language program. A key difference between APPC and IPIC autoinstall is that DFHISAIP is the default value of the URM option on a TCPIPSERVICE where IPIC is the specified protocol. Therefore, IPIC connections will be autoinstalled by default. To disable autoinstall, specify URM=NO in the TCPIPSERVICE resource definition. DFHISAIP creates an 8-character IPCONN identifier, so, if you are using IPIC connections for CICS-to-CICS communication, ensure that you specify a 4-character IPCONN name with four trailing spaces, because the REMOTESYSTEM attribute in the terminal-owning region reads the first four characters only of the IPCONN.

If the default user program is not adequate for your purposes, you can write a customized version of the default program, or create your own program to provide enhanced function. CICS supplies the source code of the default program in

several programming languages; see “Sample autoinstall user programs for IPIC connections (IPCONN)” on page 541.

Recovery and restart

Autoinstalled IPCONN resources are cataloged by CICS, for recovery at an emergency restart only. They are not recovered at a warm restart.

Autoinstall user program at INSTALL

When the autoinstall user program is invoked for the installation of an IPCONN resource, CICS passes it a parameter list in a communication area addressed by DFHEICAP.

The communication area at INSTALL for IPCONNs

The communications area is mapped by the assembler DSECT DFHISAIC, which is supplied in CICSTS52.CICS.SDFHSAMP.

```
ISAIC_FUNCTION         DS CL1    Function code (X'F0' for Install)
ISAIC_RESPONSE         DS CL1    Response code
                       DS CL2    Reserved
ISAIC_IPCONN           DS CL8    Name for the autoinstalled IPCONN
ISAIC_APPLID           DS CL8    The applid of remote system
ISAIC_SUGGESTED_APPLID DS CL8    Suggested applid, if isaic_applid
*                                is blank
ISAIC_NETWORKID        DS CL8    Network ID of remote system
ISAIC_TCPIPSERVICE     DS CL8    Name of the TCPIPSERVICE on which
*                                this connect flow arrived
ISAIC_TEMPLATE         DS CL8    Name of the template IPCONN
ISAIC_HOST             DS CL116  Host name of remote system
ISAIC_PORT             DS F      Call back port number of
*                                remote system
ISAIC_RECEIVECOUNT     DS F      Number of receive sessions wanted
*                                by remote system
```

Figure 59. Autoinstall user program's communications area at INSTALL. For IPCONNs.

isaic_applid (Input/Output)
An 8-character field containing the applid of the remote system trying to connect, as sent on the connect flow. The user program can change this value only if it is blank on input (which indicates that the connecting system is probably a Java client). If it is blank on output, CICS uses the “suggested applid” pointed to by the isaic_suggested_applid field.

isaic_function (Input)
A 1-character code indicating the function for which the autoinstall user program has been invoked. Contains X'F0' for install.

isaic_host (Input/Output)
A 116-character field containing the host name of the remote system, as passed in the connect flow. The autoinstall user program is allowed to modify this because it is possible that it has a better idea of what the connecting system is known as locally than does the system itself.

isaic_ipconn (Output)
An 8-character field containing the name to be used for the autoinstalled IPCONN connection. The user program must supply the name.

isaic_networkid (Input)
An 8-character field containing the network ID of the system trying to connect, as sent on the connect flow.

isaic_port (Input/Output)
The call back port number for the client. -1 means that no call back is allowed. The autoinstall user program can modify this for the same reason that it can modify the host name, unless it is -1, in which case it cannot be changed. The autoinstall user program is also not allowed to change this value to -1.

isaic_receivecount (Input)
A 4-byte binary field containing the number of receive sessions that the remote system wants to be supported by this IPCONN. (These are send sessions at the remote system end.)

isaic_response (Output)
Response code: zero means OK.

isaic_suggested_applid (Input)
An 8-character field containing a remote system applid "suggested" by CICS. If the applid of the remote system pointed to by isaic_applid is blank, CICS uses a counter to generate an 8-character decimal digit name in the form "00000027".

isaic_tcpipservice (Input)
An 8-character field containing the name of the TCPIPSERVICE on which this connect flow arrived.

isaic_template (Input/Output)
An 8-character field containing the name of an installed IPCONN to be used as a template for the new IPCONN resource.

By default, this field is blank, and CICS supplies the information required to create the IPCONN resource.

The autoinstall user program can modify this field to name a template IPCONN resource. If the template IPCONN resource is out-of-service, the autoinstall request is rejected.

The autoinstall user program at DELETE

To provide symmetry of control over the autoinstall process, the autoinstall user program is invoked when an autoinstalled IPCONN resource is deleted.

Invoking the user program at DELETE enables you to reverse the processes carried out at the INSTALL event. For example, if the user program at INSTALL incremented a count of the total number of automatically installed resources, then the user program at DELETE would decrement that count.

Input to the program is by a communication area, addressed by DFHEICAP. The layout of the communication area is shown in Figure 60 on page 541.

```
ISAIC_FUNCTION         DS CL1    Function code (X'F1' for Delete)
                       DS CL3    Reserved
ISAIC_IPCONN           DS CL8    Name of the autoinstalled IPCONN
ISAIC_APPLID           DS CL8    Applid of the autoinstalled IPCONN
                       DS CL8    Reserved
ISAIC_NETWORKID        DS CL8    Network ID of the autoinstalled IPCONN
ISAIC_TCPIPSERVICE     DS CL8    Name of the TCPIPSERVICE on which
*                                this connect flow arrived
```

Figure 60. Autoinstall user program's communication area at DELETE. For IPCONNs.

isaic_function (Input)

The function for which the user program has been invoked:

X'F1' After deletion of an IPCONN.

isaic_ipconn (Input)
The name of the autoinstalled IPCONN.

isaic_applid (Input)
The applid (of the remote system) specified on the autoinstalled IPCONN.

isaic_networkid (Input)
The network ID (of the remote system) specified on the autoinstalled IPCONN.

isaic_tcpipservice (Input)
The name of the TCPIPSERVICE on which the connect flow arrived.

All fields in the DELETE communications area are input-only.

When autoinstalled IPCONNs are deleted

An autoinstalled IPCONN is always deleted when they move to the RELEASED state. A RELEASED IPCONN continues to exist only when a CICS to CICS IPCONN is restored at emergency restart, when it waits for reacquire to allow recovery processing.

Sample autoinstall user programs for IPIC connections (IPCONN)

The default user program for autoinstall of IPCONNs is an assembler-language program called DFHISAIP. The corresponding copy book that defines its communications area is DFHISAIC.

The source code of the default program, and the copy book of its communications area, are supplied in assembler-language, COBOL, PL/I, and C versions. The supplied programs and copy books, and the CICSTS52.CICS libraries in which they can be found, are summarized in Table 29.

Table 29. IPCONN autoinstall user programs and copy books

	Language	Member name	Library
Programs	Assembler	DFHISAIP	SDFHSAMP
Programs	C	DFHISDIP	SDFHSAMP
Programs	COBOL	DFHISCIP	SDFHSAMP
Programs	PL/I	DFHISPIP	SDFHSAMP
Copy books	Assembler	DFHISAIC	SDFHSAMP
Copy books	C	DFHISAIC	SDFHC370
Copy books	COBOL	DFHISAIC	SDFHCOB

Table 29. IPCONN autoinstall user programs and copy books (continued)

	Language	Member name	Library
Copy books	PL/I	DFHISAIC	SDFHPL1

You can write your own IPCONN autoinstall user program in COBOL, PL/I, C, or assembler language.

Default actions of the sample program

The role of the autoinstall user program in installing IPCONNs is to choose the IPCONN template to be used and to supply the name of the new connection. Optionally, the program can modify the values of the APPLID, HOST, and PORT attributes of the new connection from those supplied on the connect flow. All other attributes of the new IPCONN are taken from the template or the CICS-supplied default values and cannot be modified by the user program.

At INSTALL

When invoked to install an IPCONN definition, the actions taken by the supplied version of the user program are to:

1. Specify a name for the new connection by setting the `isaic_ipconn` field equal to the last 4 non-blank characters of the connecting system's applid in the `isaic_applid` field.

   If the `isaic_applid` field is blank, set its value, and the connection name, to the "suggested applid" in the `isaic_suggested_applid` field.
2. Leave all other connection attributes to assume their default values, and return.

At DELETE

When invoked to delete an IPCONN definition, the supplied version of the user program takes no action and returns immediately.

Resource definitions

CICS supplies a resource definition group called DFHISCIP that defines the supplied, default, autoinstall program, DFHISAIP.

This is included in the default CICS startup grouplist, DFHLIST. If you use a different CICS startup grouplist, ensure that you append the DFHISCIP group to it.

If you customize DFHISAIP, you may need to create your own resource definitions. When you do, ensure that you append your resource definition group to your CICS startup grouplist.

Sample autoinstall user program to support predefined connection templates

The supplied source of the user-replaceable program is a sample to illustrate a technique of customizing autoinstall of an IPCONN, such that the IPCONN name and APPLID are generated according to a template IPCONN that is previously installed.

The source of the additional IPCONN autoinstall user replaceable program is supplied in the SDFHSAMP library. The code is supplied in assembler as module

DFH$ISAI and COBOL as module DFH0ISAI. The executable load modules are supplied in the CICSTS SDFHLOAD library.

The sample is for illustrative purposes and can be tailored to suit particular requirements. The program is appropriate for use when you are connecting multiple clients from the same system, for example using WebSphere Application Server for z/OS that is running in local mode with CICS Transaction Gateway.

When the user-replaceable program is deployed, all IPIC installation requests are based on a template IPCONN that must match the name of the network ID of the partner (for CICS Transaction Gateway clients, this is the APPLID qualifier). Connection requests are accepted only if the APPLID of the partner matches the APPLID value that is specified in the template IPCONN.

INSTALL

The user-replaceable program provides the following function at INSTALL.

Connection requests are rejected in the following circumstances:

- The network ID of the partner does not match the name of an installed IPCONN template.
- The APPLID of the partner is a different length to the APPLID value supplied in the IPCONN template.
- The APPLID of the partner does not match the APPLID characters that are supplied in the IPCONN template.
- The IP address of the partner does not match the HOST value, if it is defined in the IPCONN template.
- The maximum number of autoinstalled IPCONN names, which are composed of the partner APPLID followed by the count suffix, is exceeded.

If the connection request is accepted, the value of the autoinstalled IPCONN, and the APPLID it specifies, are dynamically generated using the partner APPLID followed by a unique integer suffix. The suffix is generated from the value of a count that the sample user replaceable module maintains for each template IPCONN. The current value of the count is recorded in a CICS temporary storage element. For example, if the APPLID supplied is seven characters long (for example CTGSYST), a one-character count is appended to provide the full eight character IPCONN name (CTGSYST1). In this way, a maximum of nine clients are allowed to connect from the same partner.

DELETE

The user-replaceable program provides the following function at DELETE.

When invoked to delete an IPCONN definition, the supplied version of the user program takes no action and returns immediately.

Chapter 16. Writing a program to control autoinstall of shipped terminals

You can write a program to control the installation of shipped terminals and connections. Both the supplied autoinstall control programs, DFHZATDX and DFHZATDY, provide function to install shipped definitions of remote terminals and connections. You can base your customized control program on either DFHZATDX or DFHZATDY.

Just as you can use an autoinstall user program in a terminal-owning region (TOR) to control the automatic installation of local terminals and connections, you can use a similar program in an application-owning region (AOR) to control the installation of shipped terminals and connections.

Installing shipped terminals and connections

Because your autoinstall control program is invoked for shipped terminals and connections, you can use it to reset the TERMINAL (or CONNECTION) attribute of a shipped definition to an **alias**, thereby avoiding conflicts with names of remote terminals, local terminals, sessions and connections already installed in the applications-owning region (AOR).

There is no need to reset the REMOTENAME attribute, which remains set to the name by which the terminal is known in the TOR; and autoinstall model names are not applicable to shipped definitions.

If the autoinstall control program selects a terminal name that clashes with the name of a local terminal, then the request is rejected and the autoinstall control program is not invoked again.

For more information about using aliases on remote definitions, see the *CICS Intercommunication Guide*.

Note: The autoinstall control program is invoked for all shipped terminals and connections, including shipped definitions of the virtual terminals used by CICS Clients.

CICS-generated aliases

The autoinstall control program is invoked once for each shipped terminal or connection definition to be installed. If CICS detects that the name on a shipped definition clashes with the name of a remote terminal, local terminal, session or connection already installed in the application-owning region (AOR), it generates an alias TERMID and passes it to the control program in field SELECTED_SHIPPED_TERMID of the communications area.

If CICS detects that there is no clash of names, it passes in SELECTED_SHIPPED_TERMID the name by which the terminal or connection is known in the TOR—that is, the value of the TERMINAL or CONNECTION attribute on the shipped definition.

Your control program can accept the passed TERMID, change it, or reject the installation of the shipped definition.

CICS-generated aliases consist of a 1-character prefix and a 3-character suffix. The prefix is always '{'. The suffix can have the values 'AAA' through '999'. That is, each character in the suffix can have the value 'A' through 'Z' or '0' through '9'. The first suffix generated by CICS has the value 'AAA'. This is followed by 'AAB', 'AAC', ... 'AAZ', 'AA0', 'AA1', and so on, up to '999'.

Each time that it needs to create an alias, CICS generates a 3-character suffix that it has not recorded as being in use. If your autoinstall control program overrides a CICS-generated TERMID, CICS does not record the suffix as being in use, and supplies the same suffix for the next alias.

Resetting the terminal identifier

When you write an autoinstall program, you must consider the algorithm which your control program uses to allocate alias TERMIDs.

You must consider the consequences of a definition being deleted by the CICS timeout delete mechanism, and subsequently being re-shipped and re-installed. You must decide whether your autoinstall program should allocate the *same* TERMID as before (which implies a file mapping the name by which the terminal is known in the TOR to the alias allocated by the AOR), or whether allocation of a different TERMID is acceptable—in which case you could use the default aliases generated by CICS. This decision may depend on several factors. For example:

- How your application programs allocate temporary storage queue names. If they derive them from the TERMID (so as to associate the queue with a particular end-user), problems of data mismatch could occur if the queue is not emptied by transaction end (possibly due to a failure), and TERMIDs are not allocated to the same terminals consistently.

  The best solution is for your application programs always to check before creating a temporary storage queue whether a queue of the same name already exists, and, if so, to delete it. This dispenses with the need for your autoinstall program to allocate TERMIDs consistently.

  However, if your application programs do not already implement this check, it may not be possible to correct them all. In this case, your autoinstall program may need to use a mapping file, as described.
- Whether your application programs record TERMIDs for later use. For example, an application might issue an EXEC CICS START TERMID command, with a time interval after which the transaction is to be initiated against the named terminal. If, during the delay interval, the terminal definition is deleted, re-shipped, and re-installed with a different local TERMID, the started transaction could fail because the TERMID no longer exists.

  If your application programs record TERMIDs in this way, your autoinstall program may need to use a mapping file.

Example

This example demonstrates how an AOR can resolve terminal identifiers from two terminal-owning regions that use the same set of terminal identifiers.

Assume that you have two terminal-owning regions, TORA and TORB, and that they use the same set of terminal identifiers, T001 through T500. TORA and TORB route transactions to the same application-owning region, AOR1. To prevent naming conflicts when terminals are shipped to AOR1, your control program in AOR1 could:

- Accept the TERMIDs allocated by TORA. That is, leave the TERMINAL attribute of the remote definition set to the same as the REMOTENAME attribute.

- Create aliases for the TERMIDs allocated by TORB. That is, reset the TERMINAL attribute of the remote definition, using a mapping file as described. For example, TERMIDs of T001 through T500 could be mapped to aliases of A001 through A500.

This solution allows two TORs using the same set of TERMIDs to access the same AOR. However, even though the aliases created in the AOR are mapped consistently to TERMIDs in the TOR, the solution does not *guarantee* that data mismatch problems cannot occur if terminals are re-shipped. This is because it relies on TERMIDs being allocated consistently *in the TOR*—that is, on specific TERMIDs always being assigned to the same physical devices.

Note: Your control program could use the correlation identifier contained in each terminal and connection definition to check whether a definition has been re-installed in the TOR—see the description of the INSTALL_SHIPPED_CORRID_PTR parameter in "The communications area at INSTALL for shipped terminals."

A better solution might be to map the terminal alias in the AOR to the **netname** of the terminal. This would at least guarantee that a specific alias always relates to the same physical device. But it would still require TERMIDs for which aliases are *not* created to be consistently allocated in the TOR.

The autoinstall control program at INSTALL

The autoinstall control program is invoked at INSTALL for:

- Local SNA LUs
- MVS consoles
- Local APPC single-session connections initiated by a CINIT
- Local APPC parallel-session connections initiated by a BIND
- Local APPC single-session connections initiated by a BIND
- Client virtual terminals
- Remote shipped terminals and connections, including shipped definitions of Client virtual terminals.

On each invocation, CICS passes a parameter list to the control program by means of a communication area addressed by DFHEICAP. The parameter list passed at INSTALL of local terminals and APPC single-session connections initiated by CINIT is described in "The communication area at INSTALL for terminals" on page 502. The parameter list passed at INSTALL of MVS consoles is described in "Autoinstall control program at INSTALL" on page 520. The parameter list passed at INSTALL of local APPC connections initiated by BIND requests is described in "The communication area at INSTALL for APPC connections" on page 529. The parameter list passed at INSTALL of Client virtual terminals is described in "The communications area at INSTALL for Client virtual terminals" on page 557. This section describes only INSTALL of shipped terminals and connections.

The communications area at INSTALL for shipped terminals

The communications area is mapped by the DSECT for the assembler version of DFHZATDX, which is supplied in CICSTS52.CICS.SDFHMAC.

```
*-------------------------------------------------------------------------*
* Remote install parameter list - Shipped definition functions  7 & 8     *
*-------------------------------------------------------------------------*
INSTALL_SHIPPED_COMMAREA        DSECT          Install Parameter List
*
INSTALL_SHIPPED_STANDARD        DS  F          Standard field
                                ORG INSTALL_SHIPPED_STANDARD
INSTALL_SHIPPED_EXIT_FUNCTION   DS  XL1        Install type
INSTALL_SHIPPED_TERM            EQU X'F7'      Install terminal
INSTALL_SHIPPED_RSE             EQU X'F8'      Install remote system entry
INSTALL_SHIPPED_EXIT_COMPONENT  DS CL2         Component ID 'ZC'
INSTALL_SHIPPED_CLASH           DS  CL1        Install clash Y/N
                                ORG ,
INSTALL_SHIPPED_NETNAME_PTR     DS  A          Pointer to netname
INSTALL_SHIPPED_SELECTED_PTR    DS  A          Pointer to return fields
INSTALL_SHIPPED_TERMID_PTR      DS  A          Pointer to incoming TERMID
INSTALL_SHIPPED_APPLID_PTR      DS  A          Pointer to applid of TOR
INSTALL_SHIPPED_SYSID_PTR       DS  A          Pointer to sysid
INSTALL_SHIPPED_CORRID_PTR      DS  A          Pointer to correlation ID
INSTALL_SHIPPED_SELECTED_PARMS  DSECT ,
                                DS  CL8        Reserved
SELECTED_SHIPPED_TERMID         DS  CL4        Selected TERMID
SELECTED_SHIPPED_RETURN_CODE    DS  CL1        Selected return code
RETURN_OK                       EQU X'00'      Accept request
REJECT                          EQU X'01'      Reject request
*
```

Figure 61. Autoinstall control program's communications area at INSTALL. For shipped terminals and connections.

INSTALL_SHIPPED_STANDARD
A fullword input field containing the following information:

INSTALL_SHIPPED_EXIT_FUNCTION
A 1-byte field that indicates the type of resource being installed. For install of remote terminals and connections the equated values are:

INSTALL_SHIPPED_TERM (X'F7')
A shipped terminal

INSTALL_SHIPPED_RSE (X'F8')
A shipped connection (remote system entry).

INSTALL_SHIPPED_EXIT_COMPONENT
A 2-byte component code, which is set to 'ZC'.

INSTALL_SHIPPED_CLASH
A 1-character input field that indicates whether the TERMID of the shipped definition is already in use in the AOR.

Y The name by which the terminal or connection is known in the TOR (the value of the TERMINAL or CONNECTION attribute on the shipped definition) is already in use in the AOR to identify an installed remote terminal or connection.

N The name by which the terminal or connection is known in the TOR is not in use in the AOR to identify a remote terminal or connection.

INSTALLED_SHIPPED_NETNAME_PTR
A fullword pointer to an 8-character input field containing the netname of the terminal or connection to be installed.

INSTALL_SHIPPED_SELECTED_PTR
A fullword pointer to the return fields. The output fields, for use by your program, are:

SELECTED_SHIPPED_TERMID
A 4-character field used to specify the name by which the remote terminal or connection is to be known to this system. If the name is less than 4 characters long, it must be padded with trailing blanks. For a list of the characters you can use in terminal names, see the *CICS Resource Definition Guide*.

On invocation, if INSTALL_SHIPPED_CLASH is set to 'N' (indicating no conflict of terminal names), SELECTED_SHIPPED_TERMID contains the same value as the field pointed to by INSTALL_SHIPPED_TERMID_PTR (the value of the TERMINAL or CONNECTION attribute on the shipped definition). If INSTALL_SHIPPED_CLASH is set to 'Y', SELECTED_SHIPPED_TERMID contains a CICS-generated alias.

Your user program can use this field to override a CICS-generated alias. For advice on choosing terminal and connection names, see "Resetting the terminal identifier" on page 546.

SELECTED_SHIPPED_RETURN_CODE
The 1-character return code field. The equated values are:

RETURN_OK (X'00')
Install the remote terminal or connection. Your user program must return this value if the resource is to be autoinstalled.

REJECT (X'01')
Do not install the remote terminal or connection. This is the default value.

INSTALL_SHIPPED_TERMID_PTR
A fullword pointer to a 4-character input field containing the name by which the terminal or connection is known in the TOR. (This is the value of the TERMINAL or CONNECTION attribute on the shipped definition.)

INSTALL_SHIPPED_APPLID_PTR
A fullword pointer to an 8-character input field containing the netname (applid) of the TOR.

INSTALL_SHIPPED_SYSID_PTR
A fullword pointer to a 4-character input field containing the name (sysid) of the connection to the TOR.

INSTALL_SHIPPED_CORRID_PTR
A fullword pointer to an 8-character input field containing the shipped definition's *correlation identifier*. A correlation identifier is a unique "instance token" that is created when a terminal or connection definition is installed and stored within the definition. Thus, if the definition is shipped to another region, the value of the token is shipped too. The correlation ID is used by CICS during attach processing, to check whether existing shipped definitions in an AOR are up-to-date, or whether they have to be deleted and reshipped because the terminal has been re-installed in the TOR. For further information about instance tokens, see the *CICS Intercommunication Guide*.

If your control program maps TOR-allocated TERMIDs to the aliases that it assigns in the AOR, by recording correlation IDs it could check whether a terminal has been re-installed in the TOR. If the terminal has been re-installed, it is possible that the TOR-allocated TERMID relates to a different physical device from that last installed under this TERMID.

Autoinstall control program at DELETE

The autoinstall control program is reinvoked when an autoinstalled resource is deleted. Invoking the user program at DELETE enables you to reverse the processes carried out at INSTALL.

The resources that can be autoinstalled are listed under "The autoinstall control program at INSTALL" on page 547.

The parameter list passed to your user program at DELETE of local terminals is described in "The autoinstall control program at DELETE" on page 509. The parameter list passed at DELETE of local APPC connections is described in "The autoinstall control program at DELETE" on page 533. The parameter list passed at DELETE of Client virtual terminals is described in "The autoinstall control program at DELETE" on page 561. This section describes only DELETE of shipped terminals and connections.

Shipped terminal and connection definitions are deleted by the timeout delete mechanism. For details of the timeout delete mechanism, see the *CICS Intercommunication Guide*.

Figure 62 shows the communications area passed to the autoinstall user program at DELETE.

```
DELETE_SHIPPED_COMMAREA         DSECT ,        Delete parameter list
DELETE_SHIPPED_STANDARD         DS  F          Standard field
DELETE_SHIPPED_EXIT_FUNCTION    DS  XL1        Delete  type
DELETE_SHIPPED_TERM             EQU X'FA'      Delete terminal
DELETE_SHIPPED_RSE              EQU X'FB'      Delete remote system entry
DELETE_SHIPPED_EXIT_COMPONENT   DS  CL2        Component ID 'ZC'
                                DS  CL1        Reserved
DELETE_SHIPPED_TERMID           DS  CL4        TERMID in TOR
DELETE_SHIPPED_APPLID           DS  CL8        Applid of TOR
DELETE_SHIPPED_LTERMID          DS  CL4        TERMID in AOR
DELETE_SHIPPED_NETNAME          DS  CL8        Netname of terminal
```

Figure 62. Autoinstall control program's communications area at DELETE. For shipped terminals and connections.

At DELETE, all fields in the communications area are input only. Fields not listed are as described for INSTALL.

DELETE_SHIPPED_EXIT_FUNCTION
A 1-byte field that indicates the type of resource being deleted. The equated values are:

DELETE_SHIPPED_TERM (X'FA')
A shipped terminal

DELETE_SHIPPED_RSE (X'FB')
A shipped connection (remote system entry).

DELETE_SHIPPED_TERMID
A 4-character field containing the identifier (TERMID) of the terminal or connection in the TOR.

DELETE_SHIPPED_APPLID
An 8-character field containing the netname (applid) of the TOR.

DELETE_SHIPPED_LTERMID
A 4-character field containing the name by which the terminal or connection is

known in the AOR. This may or may not be the same as DELETE_SHIPPED_TERMID, depending on whether an alias has been used in the AOR.

DELETE_SHIPPED_NETNAME
An 8-character field containing the netname of the terminal being deleted.

Default actions of the sample programs

When DFHZATDX or DFHZATDY is invoked at INSTALL of a shipped terminal or connection, it:

1. Updates, if necessary, the SELECTED_SHIPPED_TERMID field, so that it contains the name by which the terminal or connection is known in the TOR.

   Note:

   a. If CICS detected a conflict with a currently-installed remote TERMID, on invocation of the sample programs SELECTED_SHIPPED_TERMID contains a CICS-generated alias. The sample programs overwrite this value.

   b. If CICS detected no conflict with a currently-installed remote TERMID, on invocation of the sample programs SELECTED_SHIPPED_TERMID contains the value of the TERMINAL attribute on the shipped definition (the value pointed to by INSTALL_SHIPPED_TERMID_PTR). The sample programs accept this value.

2. Permits the remote definition to be installed by setting the return code field to RETURN_OK, and returning.

When DFHZATDX or DFHZATDY is invoked at DELETE of a shipped terminal or connection, it takes no action and returns.

Chapter 17. Writing a program to control autoinstall of virtual terminals

You can write a program to control the installation of virtual terminals. Virtual terminals are used by the External Presentation Interface (EPI) and terminal emulator functions of CICS clients and the CICS Link3270 bridge. Both the supplied autoinstall control programs, DFHZATDX and DFHZATDY, provide function to install definitions of virtual terminals. You can base your customized control program on either DFHZATDX or DFHZATDY.

In a bridged environment, the virtual terminals, known as *bridge facilities*, replace the real 3270 that is the principal facility of a 3270 transaction.

How Client virtual terminals are autoinstalled

Client virtual terminals are defined to CICS as remote 3270 datastream devices.

Autoinstall models

The autoinstall control program cannot choose a different autoinstall model. The autoinstall model used to install a virtual terminal is determined using the following sequence:

1. **For EPI programs:**From the **DevType** parameter of the **CICS_EpiAddTerminal** function, if specified by the Client EPI program. (For details of EPI calls, see the *CICS Family: Client/Server Programming* manual.)

   For the Client terminal emulator: From the **/m** parameter of the **cicsterm** command used to start the emulator, if specified by the workstation user. (For details of the **cicsterm** command, see the *CICS Clients: Administration* manual.)

   Note: Any autoinstall models specified by Clients must, of course, be defined to CICS. However, because z/OS Communications Server definitions are not required for Client virtual terminals, there is no need to create matching entries in the z/OS Communications Server LOGMODE table.
2. The CICS-supplied autoinstall model, DFHLU2.

Terminal identifiers

The terminal identifier (TERMID) passed to the CICS autoinstall function at install of a virtual terminal is determined using the following sequence:

1. **For EPI programs:** From the **NetName** parameter of the **CICS_EpiAddTerminal** function, if specified by the Client EPI program.

   For the Client terminal emulator: From the **/n** parameter of the **cicsterm** command used to start the emulator, if specified by the workstation user.
2. A name generated automatically by CICS.

   TERMIDs generated by CICS for Client terminals consist of a 1-character prefix and a 3-character suffix. The default prefix is '\', but you can specify a different prefix using the VTPREFIX system initialization parameter. The suffix can have the values 'AAA' through '999'. That is, each character in the suffix can have the value 'A' through 'Z' or '0' through '9'. The first suffix generated by CICS has the value 'AAA'. This is followed by 'AAB', 'AAC', ... 'AAZ', 'AA0', 'AA1', and so on, up to '999'.

Each time a Client virtual terminal is autoinstalled, CICS generates a 3-character suffix that it has not recorded as being in use.

Note: By specifying a prefix, you can ensure that the TERMIDs of Client terminals autoinstalled on this system are unique in your transaction routing network. This prevents the conflicts that could occur if two or more regions ship definitions of virtual terminals to the same application-owning region (AOR).

For brevity, the name specified by the Client or the generated **VTPREFIX** name is the *supplied name*. The Client always knows the virtual terminal by the supplied name. However, your autoinstall control program can allocate an alias, by which the virtual terminal is known to CICS.

If the CICS autoinstall function detects that the supplied name clashes with the name of a remote terminal or connection already installed on this region, it generates an alias TERMID. CICS generates alias TERMIDs for virtual terminals in the same way as it generates aliases for shipped terminals—see "CICS-generated aliases" on page 545.

Note: If the supplied name clashes with the name of a local terminal or connection, the install of the virtual terminal is rejected, and the autoinstall control program is not invoked.

The autoinstall control program is invoked once for each virtual terminal definition to be installed. When it is invoked, field INSTALL_SHIPPED_TERMID_PTR of the communications area points to the supplied TERMID. Field SELECTED_SHIPPED_TERMID contains either the supplied TERMID, or a generated alias, depending on whether a clash of names has been detected.

Your control program can accept the TERMID passed in SELECTED_SHIPPED_TERMID, change it, or reject the installation of the virtual terminal.

Why override TERMIDs?

Why might you want to create an alias for the supplied TERMID (or, in the case of a clash of names, to override the alias generated by CICS)? You may not need to; it may depend on the way in which your server programs are written. By "server programs" we mean both the transaction programs started by Client EPI programs, and those started from the Client terminal emulator.

Overriding CICS-generated TERMIDs

If you are using CICS-generated TERMIDs and have specified a different prefix, reserved for virtual terminals, on each region on which Client terminals can be installed, there should be no clash of names, either in the regions in which the virtual terminals are installed, or when different regions ship Client definitions to the same AOR.

However, if you are using CICS-generated TERMIDs, your server programs must not rely on TERMIDs being allocated consistently to particular Client terminals.

A Client terminal can be deleted by a Client sending a **CICS_EpiDelTerminal** request, by an end user shutting down a Client terminal emulator or the Client itself, or if a connection failure occurs. When it is reinstalled, CICS does not necessarily generate the same TERMID as it had previously. This could create problems if, for example:

- Your server programs derive temporary storage queue names from the TERMID (to associate each queue with a particular end user). Problems of data mismatch could occur if the queue is not deleted by transaction end (possibly due to a failure).

  The best solution is for your application programs always to check before creating a temporary storage queue whether a queue of the same name already exists, and, if so, to delete it. However, if you have a large number of server applications, it may not be possible to check or change them all.
- Your server programs record TERMIDs for later use. For example, an application might issue an EXEC CICS START TERMID command, with a time interval after which the transaction is to be initiated against the named terminal. If, during the delay interval, the virtual terminal is deleted, and re-installed with a different TERMID, the started transaction could fail because the TERMID no longer exists.

If your server programs cannot be rewritten, it may be necessary for your autoinstall control program to create aliases for the CICS-generated TERMIDs. It could, for example, use a mapping file to relate particular aliases to particular Client workstations (identified by connection name).

If your server programs are located on a back-end AOR, the autoinstall control program is invoked in the AOR when a virtual terminal is shipped in, just as for any other shipped definition. It can, if necessary, allocate an alias terminal identifier to the shipped definition. (For details of writing a control program to install shipped definitions, see Chapter 16, “Writing a program to control autoinstall of shipped terminals,” on page 545.)

Overriding Client-specified TERMIDs

If TERMIDs are always nominated, in a consistent way, by your Client EPI programs, the problem of data mismatch due to server programs recording TERMIDs should not occur.

However, Client-specified TERMIDs could clash with non-Client remote TERMIDs; or, if several Clients are attached to the same CICS system, with each other. If this occurs in the region on which the CTIN transaction runs, for consistency your autoinstall control program may need to allocate alias TERMIDs, rather than relying on the aliases provided by CICS. (That is, it may need to relate particular TERMIDs to particular Client workstations, as previously described.)

If a name clash occurs in an AOR, the autoinstall control program is invoked in the AOR. It can resolve the conflict by allocating an alias terminal identifier to the shipped definition.

How bridge facility virtual terminals are autoinstalled

Bridge facility virtual terminals are defined as LU2 devices. They are created by the 3270 bridge mechanism to support the execution of a CICS 3270 application in a bridged environment, where all terminal interaction is intercepted and passed to the 3270 bridge mechanism.

The 3270 bridge mechanism uses a model 3270 terminal definition (***facilitylike***) to build the bridge facility, creating both an eight-byte token to identify it and a four-character terminal identifier, which is used as both TERMID and NETNAME.

For bridge facilities created with the START BREXIT command, the token and name are unique within the region creating the bridge facility, and the TERMID takes the form *}AAA*, where AAA is an alphabetic sequence that ascends serially.

For bridge facilities created by the link3270 bridge, the token and name are unique across the CICSplex, and the TERMID is of the form *AAA}*. Uniqueness is achieved by using a shared file to control allocation of names.

Bridge facility terminal names and netnames are normally allocated dynamically by the bridge mechanism, but if the **AIBRIDGE** system initialization parameter is set to YES, the terminal autoinstall control program is called and can be used to assign installation specific names.

Using the terminal autoinstall control program for bridge facilities

If you specify AIBRIDGE (YES), then the autoinstall control program is called when a bridge facility is allocated or deleted.

The autoinstall control program is passed a parameter list (the communications area) described in "The communications area at INSTALL for bridge facility virtual terminals" on page 559 and "The communications area at DELETE for bridge facility virtual terminals" on page 562. This indicates whether the program was called for a Link3270 or a START bridge facility.

Installation specific terminal names can be allocated in one of two ways:

- Names can be defined by the client program and passed on the initial Link3270 request. The autoinstall control program can then allow, change or reject these names.
- Names can be defined by the autoinstall control program

The names suggested by the client program are passed in the communications area. The autoinstall control program can also use EXEC CICS ASSIGN USERID to obtain the USERID and use this to validate the suggested TERMID and NETNAME.

The client defined TERMID and NETNAME fields can also be used to pass some installation specific data to the autoinstall control program, to be used to generate the required names.

Autoinstall of a START bridge facility

Apart from the information contained in the communications area, you can obtain the following information:

- The USERID of the first transaction in a pseudoconversation can be obtained using **EXEC CICS ASSIGN USERID**.
- The TRANSID of the first transaction in a pseudoconversation can be obtained from EIBTRNID.

The autoinstall control program can use the USERID and TRANSID values to derive new TERMID and NETNAME values and return them in the communications area.

Bridge facility name uniqueness

Some applications use the termid to allocate a unique resource. This relies on the name being unique within the CICSPlex. Bridge facility names have the same namespace as termids. However, CICS is unable to ensure that the bridge facility name returned by the autoinstall control program is not the same as a termid somewhere in the CICSPlex. Neither the termid nor netname returned by the autoinstall control program are validated.

The autoinstall control program at INSTALL

The autoinstall control program is invoked at INSTALL for:

- Local SNA LUs
- MVS consoles
- Local APPC single-session connections initiated by a CINIT
- Local APPC parallel-session connections initiated by a BIND
- Local APPC single-session connections initiated by a BIND
- Client virtual terminals
- Bridge facility virtual terminals
- Remote shipped terminals and connections (including shipped definitions of Client virtual terminals).

On each invocation, CICS passes a parameter list to the control program by means of a communication area addressed by DFHEICAP. The parameter list passed at INSTALL of local terminals and APPC single-session connections initiated by CINIT is described in "The communication area at INSTALL for terminals" on page 502. The parameter list passed at INSTALL of local APPC connections initiated by BIND requests is described in "The communication area at INSTALL for APPC connections" on page 529. The parameter list passed at INSTALL of MVS consoles is described in "Autoinstall control program at INSTALL" on page 520. The parameter list passed at INSTALL of shipped terminals and connections is described in "The communications area at INSTALL for shipped terminals" on page 547. This section describes only parameters passed at INSTALL of Client virtual terminals, in "The communications area at INSTALL for Client virtual terminals," and of bridge facilities in "The communications area at INSTALL for bridge facility virtual terminals" on page 559.

The communications area at INSTALL for Client virtual terminals

The communications area is mapped by the DSECT for the assembler version of DFHZATDX or DFHZATDY, which are supplied in CICSTS52.CICS.SDFHMAC.

Note: The communications area for INSTALL of virtual terminals is the same as that for INSTALL of shipped terminals and connections—that is why the field names contain the word "SHIPPED".

```
*-------------------------------------------------------------------------*
* Remote install parameter list - Client virtual terminal function   9     *
*-------------------------------------------------------------------------*
INSTALL_SHIPPED_COMMAREA          DSECT          Install Parameter List
*
INSTALL_SHIPPED_STANDARD          DS  F          Standard field
                                  ORG INSTALL_SHIPPED_STANDARD
INSTALL_SHIPPED_EXIT_FUNCTION     DS  XL1        Install type
INSTALL_SHIPPED_TERM              EQU X'F9'      Install virtual terminal
INSTALL_SHIPPED_EXIT_COMPONENT    DS CL2         Component ID 'ZC'
INSTALL_SHIPPED_CLASH             DS  CL1        Install clash Y/N
                                  ORG ,
INSTALL_SHIPPED_NETNAME_PTR       DS  A          Pointer to netname of Client
INSTALL_SHIPPED_SELECTED_PTR      DS  A          Pointer to return fields
INSTALL_SHIPPED_TERMID_PTR        DS  A          Pointer to incoming TERMID
INSTALL_SHIPPED_APPLID_PTR        DS  A          Pointer to applid of Client
INSTALL_SHIPPED_SYSID_PTR         DS  A          Pointer to sysid of Client
INSTALL_SHIPPED_CORRID_PTR        DS  A          Pointer to correlation ID
INSTALL_SHIPPED_SELECTED_PARMS    DSECT ,
                                  DS  CL8        Reserved
SELECTED_SHIPPED_TERMID           DS  CL4        Selected TERMID
                                  DS  CL4        Reserved
                                  DS  CL4        Reserved
SELECTED_SHIPPED_RETURN_CODE      DS  CL1        Selected return code
RETURN_OK                         EQU X'00'      Accept request
REJECT                            EQU X'01'      Reject request
*
```

Figure 63. Autoinstall control program's communications area at INSTALL

INSTALL_SHIPPED_STANDARD
A fullword input field containing the following information:

INSTALL_SHIPPED_EXIT_FUNCTION
A 1-byte field that indicates the type of resource being installed. For install of Client virtual terminals the equated value is INSTALL_SHIPPED_TERM (X'F7').

INSTALL_SHIPPED_EXIT_COMPONENT
A 2-byte component code, which is set to 'ZC'.

INSTALL_SHIPPED_CLASH
A 1-character input field that indicates whether the supplied TERMID is already in use in this region.

Y The name passed to the CICS autoinstall function is already in use in this region to identify an installed remote terminal or connection.

N The name passed to the CICS autoinstall function is not already in use in this region to identify a remote terminal or connection.

INSTALL_SHIPPED_NETNAME_PTR
A fullword pointer to an 8-character field containing the netname of the Client workstation. This field contains the same value as the field pointed to by INSTALL_SHIPPED_APPLID_PTR.

INSTALL_SHIPPED_SELECTED_PTR
A fullword pointer to the return fields. The output fields, for use by your program, are:

SELECTED_SHIPPED_TERMID
A 4-character field used to specify the name by which the virtual terminal will be known to CICS. If the name is less than 4 characters long, it must be padded with trailing blanks. For a list of the characters you can use in terminal names, see the *CICS Resource Definition Guide*.

On invocation, if INSTALL_SHIPPED_CLASH is set to 'N' (indicating no conflict of terminal names), SELECTED_SHIPPED_TERMID contains the same value as the field pointed to by INSTALL_SHIPPED_TERMID_PTR (the supplied name). If INSTALL_SHIPPED_CLASH is set to 'Y', SELECTED_SHIPPED_TERMID contains a CICS-generated alias.

Your user program can override the suggested name.

SELECTED_SHIPPED_RETURN_CODE
The 1-character return code field. The equated values are:

RETURN_OK (X'00')
Install the virtual terminal. This is the default value. Your user program must return this value if the resource is to be autoinstalled.

REJECT (X'01')
Do not install the virtual terminal.

INSTALL_SHIPPED_TERMID_PTR
A fullword pointer to a 4-character input field containing the TERMID passed to the CICS autoinstall function (that is, the supplied name).

INSTALL_SHIPPED_APPLID_PTR
A fullword pointer to an 8-character input field containing the netname (applid) of the Client workstation.

INSTALL_SHIPPED_SYSID_PTR
A fullword pointer to a 4-character input field containing the name (sysid) of the connection to the Client workstation.

INSTALL_SHIPPED_CORRID_PTR
A fullword pointer to an 8-character input field that is not used for install of virtual terminals.

The communications area at INSTALL for bridge facility virtual terminals

The communications area is mapped by the DSECT for the assembler version of DFHZATDX or DFHZATDY, which are supplied in CICSTS52.CICS.SDFHMAC.

```
 ---------------------------------------------------------------------
* Install Bridge Facility                       - Function 15 & 17
*---------------------------------------------------------------------*
INSTALL_BRFAC_COMMAREA       DSECT       Install Parameter List
INSTALL_BRFAC_STANDARD       DS  F       Standard field
                             ORG INSTALL_BRFAC_STANDARD
INSTALL_BRFAC_EXIT_FUNCTION  DS  XL1     Install type
INSTALL_LINK_BRFAC            EQU X'0F' Install Link Brfacility
INSTALL_START_BRFAC           EQU X'11' Install Start Brfacility
INSTALL_BRFAC_EXIT_COMPONENT DS CL2      Component ID 'BR'
                             DS  CL1     Reserved
                             ORG ,
INSTALL_BRFAC_NETNAME_PTR    DS  A       Pointer to input netname
INSTALL_BRFAC_SELECTED_PTR   DS  A       Pointer to return fields
INSTALL_BRFAC_TERMID_PTR     DS  A       Pointer to input termid
                             DS  A       Reserved
                             DS  A       Reserved
                             DS  A       Reserved
INSTALL_BRFAC_SELECTED_PARMS DSECT ,
                             DS  CL8     Reserved
SELECTED_BRFAC_TERMID        DS  CL4     Selected termid
SELECTED_BRFAC_RETURN_CODE   DS  B       Selected return
SELECTED_BRFAC_NETNAME  DS  CL8     Selected netname
*
*---------------------------------------------------------------------*
```

Figure 64. Autoinstall control program's communications area at INSTALL

INSTALL_BRFAC_STANDARD
A fullword input field containing the following information:

INSTALL_BRFAC_EXIT_FUNCTION
A 1-byte field that indicates the type of resource being installed. For install of bridge facility virtual terminals. The equated values are:

INSTALL_LINK_BRFAC (X'0F')
The autoinstall program was called during installation of a bridge facility to be used by the link3270 bridge.

INSTALL_START_BRFAC (X'11')
The autoinstall program was called during installation of a bridge facility to be used by the START bridge.

INSTALL_BRFAC_EXIT_COMPONENT
A 2-byte component code, which is set to 'BR'.

INSTALL_BRFAC_NETNAME_PTR
A fullword pointer to an 8-character field containing the netname of the bridge facility. This is either the value specified by the client or the value generated by CICS if the client specifies BRIHNN-DEFAULT (the default value).

INSTALL_BRFAC_SELECTED_PTR
A fullword pointer to the return fields. The output fields, for use by your program, are:

SELECTED_BRFAC_TERMID
A 4-character field used to specify the name by which the virtual terminal will be known to CICS. If the name is less than 4 characters long, it must be padded with trailing blanks. For a list of the characters you can use in terminal names, see the *CICS Resource Definition Guide*. You can copy the name in INSTALL_BRFAC_TERMID_PTR, or set a new value.

SELECTED_BRFAC_RETURN_CODE
The 1-character return code field. The equated values are:

RETURN_OK (X'00')
Install the virtual terminal. This is the default value. Your user program must return this value if the resource is to be autoinstalled.

REJECT (X'01')
Do not install the virtual terminal.

SELECTED_BRFAC_NETNAME
An 8-character field used to specify the netname of the bridge facility. If the name is less than 8 characters long, it must be padded with trailing blanks. You can copy the name in INSTALL_BRFAC_NETNAME_PTR, or set a new value.

INSTALL_BRFAC_TERMID_PTR
A fullword pointer to a 4-character input field containing the TERMID passed to the CICS autoinstall function (that is, the supplied name).

The autoinstall control program at DELETE

The autoinstall control program is reinvoked when an autoinstalled resource is deleted. Invoking the user program at DELETE enables you to reverse the processes carried out at INSTALL.

The resources that can be autoinstalled are listed under "The autoinstall control program at INSTALL" on page 557.

The parameter list passed to your user program at DELETE of local terminals is described in "The autoinstall control program at DELETE" on page 509. The parameter list passed at DELETE of local APPC connections is described in "The autoinstall control program at DELETE" on page 533. The parameter list passed at DELETE of shipped definitions is described in "Autoinstall control program at DELETE" on page 550. This section describes DELETE of Client virtual terminals at "The communications area at DELETE for Client virtual terminals" and bridge facility virtual terminals at "The communications area at DELETE for bridge facility virtual terminals" on page 562.

Shipped terminal and connection definitions are deleted by the CICS timeout delete mechanism. For details of the timeout delete mechanism, see Efficient deletion of shipped terminal definitions, in the *CICS Intercommunication Guide*.

The communications area at DELETE for Client virtual terminals

The communications area passed to the autoinstall user program at DELETE.

```
DELETE_SHIPPED_COMMAREA         DSECT ,        Delete parameter list
DELETE_SHIPPED_STANDARD         DS  F          Standard field
DELETE_SHIPPED_EXIT_FUNCTION    DS  XL1        Delete  type
DELETE_SHIPPED_TERM             EQU X'FC'      Delete virtual terminal
DELETE_SHIPPED_EXIT_COMPONENT   DS  CL2        Component ID 'ZC'
                                DS  CL1        Reserved
DELETE_SHIPPED_TERMID           DS  CL4        TERMID
DELETE_SHIPPED_APPLID           DS  CL8        Applid of Client workstation
DELETE_SHIPPED_LTERMID          DS  CL4        TERMID in this region
DELETE_SHIPPED_NETNAME          DS  CL8        Netname of Client workstation
```

Figure 65. Communications area of the autoinstall control program at DELETE. For Client virtual terminals.

At DELETE, all fields in the communications area are input only. Fields not listed in the following are as described for INSTALL.

DELETE_SHIPPED_EXIT_FUNCTION
A 1- byte field that indicates the type of resource being deleted. The equated value for Client virtual terminals is DELETE_SHIPPED_TERM (X'FC').

Note: A value of X'F1' represents the deletion of a local terminal, or an APPC single-session device that was autoinstalled using a CINIT request—see "The autoinstall control program at DELETE" on page 509. A value of X'F5' or X'F6' represents the deletion of an APPC connection that was installed by a BIND request—see "The autoinstall control program at DELETE" on page 533. A value of X'FA' or X'FB' represents the deletion of a shipped terminal or connection—see Figure 62 on page 550.

DELETE_SHIPPED_TERMID
A 4- character field containing the name by which the virtual terminal is known to the Client.

DELETE_SHIPPED_APPLID
An 8- character field containing the netname (applid) of the Client workstation.

DELETE_SHIPPED_LTERMID
A 4- character field containing the name by which the virtual terminal is known in this region. This might or might not be the same as the value in DELETE_SHIPPED_TERMID, depending on whether an alias was used at install.

DELETE_SHIPPED_NETNAME
An 8- character field containing the netname of the Client workstation. This field contains the same value as DELETE_SHIPPED_APPLID.

The communications area at DELETE for bridge facility virtual terminals

The communications area passed to the autoinstall user program at DELETE.

```
*
*-------------------------------------------------------------------*
* Delete  Bridge Facility                      - Function 16        *
*-------------------------------------------------------------------*
DELETE_BRFAC_COMMAREA       DSECT ,   Delete parameter list
DELETE_BRFAC_STANDARD       DS  F     Standard field
                            ORG DELETE_BRFAC_STANDARD
DELETE_BRFAC_EXIT_FUNCTION  DS  XL1   Delete  type
DELETE_LINK_BRFAC             EQU X'10' Delete Link Brfacility
DELETE_START_BRFAC            EQU X'12' Delete Start Brfacility
DELETE_BRFAC_EXIT_COMPONENT DS  CL2   Component ID 'BR'
                            DS  CL1   Reserved
DELETE_BRFAC_TERMID         DS  CL4   Termid
                            DS  CL8   Reserved
                            DS  CL4   Reserved
DELETE_BRFAC_NETNAME   DS  CL8   Netname of terminal
```

Figure 66. Autoinstall control program's communications area at DELETE. For bridge facility virtual terminals.

At DELETE, all fields in the communications area are input only. Fields not in the following list are as described for INSTALL.

DELETE_BRFAC_EXIT_FUNCTION
A 1-byte field that indicates the type of resource being deleted. For deletion of bridge facility virtual terminals. The equated values are:

INSTALL_LINK_BRFAC (X'10')
The autoinstall program was called during installation of a bridge facility to be used by the link3270 bridge.

INSTALL_START_BRFAC (X'12')
The autoinstall program was called during installation of a bridge facility to be used by the START bridge.

DELETE_BRFAC_EXIT COMPONENT
A 2-byte component code, which is set to 'BR'

DELETE_BRFAC_TERMID
A 4-character field containing the bridge facility name.

DELETE_BRFAC_NETNAME
An 8-character field containing the netname of the bridge facility.

Default actions of the sample programs

When DFHZATDX or DFHZATDY is invoked at INSTALL of a Client virtual terminal, it:

1. Accepts the terminal name placed by CICS in SELECTED_SHIPPED_TERMID.

   If CICS detected no conflict with a currently-installed remote TERMID, SELECTED_SHIPPED_TERMID contains the value pointed to by INSTALL_SHIPPED_TERMID_PTR (that is, the name specified by the Client, or the "VTPREFIX" name generated by CICS).

   If CICS detected a conflict with a currently-installed remote TERMID, SELECTED_SHIPPED_TERMID contains a CICS-generated alias.
2. Permits the remote definition to be installed by leaving the return code field set to its default value of RETURN_OK, and returning.

When DFHZATDX or DFHZATDY is invoked at DELETE of a Client virtual terminal, it takes no action and returns.

Chapter 18. Writing a program to control autoinstall of programs

You can write a program to control the automatic installation of programs, mapsets, and partitionsets. Program autoinstallation means the automatic autoinstallation of all three program types, unless otherwise specified.

Autoinstalling programs—preliminary considerations

As well as terminals, IPCONN resources, and APPC connections, you can autoinstall programs, mapsets, and partitionsets. If the autoinstall program function is enabled, and an implicit or explicit load request is issued for a previously undefined program, mapset, or partitionset, CICS dynamically creates a definition, and installs and catalogs it, as appropriate.

An implicit or explicit load occurs when:

- CICS starts a transaction.
- An application program issues one of the following commands:
  - **EXEC CICS LINK** - see "Autoinstall programs started by **EXEC CICS LINK** commands" on page 566
  - **EXEC CICS XCTL**
  - **EXEC CICS LOAD**
  - **EXEC CICS ENABLE** (for a global user exit, or task-related user exit, program)
  - **EXEC CICS RECEIVE** or **SEND MAP**
  - **EXEC CICS SEND PARTNSET**
  - **EXEC CICS RECEIVE PARTN**
  - A dynamic COBOL call
- A program abend occurs, and CICS transfers control to the program named on an **EXEC CICS HANDLE ABEND** command.
- CICS calls any user-replaceable program other than the program or terminal autoinstall program.
- A program is named in the PLTPI or PLTSD list.

Autoinstall model definitions

Program autoinstall uses *model definitions*, together with a user-replaceable control program, to create explicit definitions for resources that need to be autoinstalled.

The purpose of a model is to provide CICS with a definition that can be used for all programs with the same properties. CICS calls the autoinstall control program with a parameter list that includes the name of a CICS-supplied, default model definition appropriate to the program type (program, mapset, or partitionset). Your autoinstall control program can accept the default model, or specify another (any installed program definition can be used as a model). It can also specify explicitly any properties that are unique to a program, thus overriding those specified on the model definition. It can specify that a local or a remote definition should be installed.

On return from the control program, CICS creates a resource definition from the model and properties returned in the parameter list.

For CICS programs, mapsets, or partitionsets (that is, for any objects that begin with the letters "DFH"), CICS uses the default model definitions, but does **not** call the user-replaceable autoinstall control program. If you have your own autoinstall control program, you cannot use it to change the resource definitions for objects that begin with the letters "DFH".

Autoinstall programs started by `EXEC CICS LINK` commands

Autoinstall programs have a relationship to the dynamic routing program. When the autoinstall control program is started because there is no installed definition of a program named on an **`EXEC CICS LINK`** command without a SYSID, the autoinstall control program can install a different definition depending on the circumstances.

The autoinstall control program can install the following definitions:

A local definition of the server program
CICS runs the server program on the local region.

A definition that specifies REMOTESYSTEM(*remote_region*) and DYNAMIC(NO)
CICS ships the LINK request to the remote region.

A definition that specifies DYNAMIC(YES)
CICS starts the dynamic routing program to route the LINK request.

Note: The DYNAMIC attribute takes precedence over the REMOTESYSTEM attribute. Therefore, a definition that specifies both REMOTESYSTEM(*remote_region*) and DYNAMIC(YES) defines the program as dynamic, instead of located on a particular remote region. In this case, the REMOTESYSTEM attribute names the default server region passed to the dynamic routing program.

No definition of the server program
CICS starts the dynamic routing program to route the LINK request.

Note: This situation assumes that the autoinstall control program does not install a definition. If no definition is installed because autoinstall fails, the dynamic routing program is not started.

Autoinstall processing of mapsets

Table 30 shows the differences in mapset processing between CICS regions with program autoinstall active and inactive.

Table 30. Differences in mapset processing between autoinstall and non-autoinstall

Program autoinstall INACTIVE	Program autoinstall ACTIVE
CSD definition is required. CICS attempts to load a referenced mapset with a suffix. If this fails, CICS tries an unsuffixed version. If that is unsuccessful, abend APCT is issued.	CSD definition is not required. Using autoinstall, CICS attempts to load the referenced suffixed mapset or partitionset, then the unsuffixed one. (In each case, a definition is autoinstalled.) The transaction requesting the resource abends only if no version of the resource exists in the library, either suffixed or unsuffixed. If the suffixed mapset was not found in the library, the definition is marked ‘not loadable’.

System autoinstall

Some programs are autoinstalled automatically (if they have not been statically defined) by the CICS *system autoinstall* function, which does not require model definitions or the support of the autoinstall control program.

Programs in this category include:

- First phase program list table post initialization (PLTPI) programs (that is, PLTPI programs that are defined before the PLT table delimiter DFHDELIM).
- Second phase program list table shutdown (PLTSD) programs (that is, PLTSD programs that are defined after the PLT table delimiter DFHDELIM).

Note: PLTPI programs that are defined after DFHDELIM, and PLTSD programs that are defined before DFHDELIM, are treated like any other user programs—they are eligible for program autoinstall.

Benefits of autoinstalling programs

Program autoinstall reduces system administration, virtual storage usage, and, potentially, restart times.

Reduced system administration costs

Without autoinstall, you have to define all new programs, mapsets, and partitionsets to CICS before they can be used. Autoinstall eliminates this requirement, enabling these resources to be used without prior definition. Furthermore, the need to maintain predefined definitions also disappears, leading to a significant saving in system administration effort.

Saving in virtual storage

There is a saving in virtual storage within the CICS address space, as the definitions of autoinstalled resources do not occupy table space until they are generated.

Faster startup times

Warm and emergency starts

When you use program autoinstall, the restart time depends upon whether you are using program autoinstall with or without cataloging.

If you are using program autoinstall with cataloging, restart times are similar to those of restarting a CICS region that is not using program autoinstall. This is because, in both cases, resource definitions are reinstalled from the catalog during the restart. The definitions after the restart are those that existed before the system was terminated.

If you are using autoinstall *without cataloging*, CICS restart times are improved because CICS does not install definitions from the CICS global catalog. Instead, definitions are autoinstalled as required whenever programs, mapsets, and partitionsets are referenced following the restart.

See the Troubleshooting for recovery processing in Troubleshooting and support for information on cataloging.

Initial and cold starts

Startup times are faster than for a region that does not use program autoinstall, because program definitions are installed singly, as required, rather than all together at startup.

Configuring autoinstall for programs

To automatically install programs, mapsets, and partitionsets, you must configure CICS to use an autoinstallation program and install the required resources.

About this task

CICS supplies an autoinstallation program called DFHPGADX, but you can write your own customized version if required.

Procedure

1. Write a customized version of the autoinstall control program for programs, DFHPGADX, unless the supplied version is entirely suitable for your purposes.
2. Specify the name of your control program on the **PGAIEXIT** system initialization parameter, or on a **SET SYSTEM PROGAUTOEXIT** command. The default name is DFHPGADX. For more information on this parameter, see PGAIEXIT system initialization parameter in Reference -> System definition.
3. Make program autoinstall active by specifying ACTIVE on the **PGAIPGM** system initialization parameter, or by issuing a **SET SYSTEM PROGAUTOINST(AUTOACTIVE)** command. For more information on this parameter, see PGAIPGM system initialization parameter in Reference -> System definition.
4. Specify whether you want autoinstalled program definitions to be recorded on the CICS global catalog. You can use the **PGAICTLG** system initialization parameter or a **SET SYSTEM PROGAUTOCTLG** command. For more information on this parameter, see PGAICTLG system initialization parameter in Reference -> System definition.
5. Include the DFHPGAIP resource definition group in your CICS startup group list. DFHPGAIP is already included in the supplied startup list, DFHLIST. It contains the default program, mapset, and partitionset model definitions passed to the autoinstall control program, and a definition of DFHPGADX. You might have to amend the definition for DFHPGADX.
6. Create any additional program, mapset, and partitionset model definitions that you need, and add this group to your startup group list.
7. If you want to log messages associated with program autoinstall, define the CSPL transient data (TD) queue.

Results

You have successfully configured autoinstallation for programs, mapsets, and partitionsets.

The autoinstall control program at INSTALL

On invocation, CICS passes a parameter list to the autoinstall control program by means of a communication area addressed by DFHEICAP. The communications area is mapped by a copybook that is supplied in each of the languages supported by CICS.

The assembler form of the parameter list is as follows:

PGAC_PROGRAM
passes the 8-byte name of the object to be autoinstalled. This is an input-only field, which your user-replaceable program must not alter.

PGAC_MODULE_TYPE
passes a 1-byte indicator of the type of object to be installed. The equated values are:

PGAC_TYPE_PROGRAM
A program

PGAC_TYPE_MAPSET
A mapset

PGAC_TYPE_PARTITIONSET
A partitionset.

This is an input-only field, which your user-replaceable program must not alter.

PGAC_MODEL_NAME
allows your control program to specify the 8-byte autoinstall model name to be used. If you do not set this field, CICS uses the default model name for the type of object:

DFHPGAPG
For a program

DFHPGAMP
For a mapset

DFHPGAPT
For a partitionset.

PGAC_LANGUAGE
allows your control program to specify, in a 1-byte field, the language of the program to be autoinstalled. The equated values are:

PGAC_ASSEMBLER
Assembler

PGAC_COBOL
COBOL

PGAC_C370
C

PGAC_LE370
Language Environment

PGAC_PLI
PL/I.

If you do not set this field, the autoinstall routine uses the language defined in the model, if one is specified. However, when control is passed to the program, CICS determines the language from the program itself, and overrides any specification provided.

You should not need to specify the language of executable programs that have been translated using the EXEC CICS translator before compiling.

PGAC_CEDF_STATUS
allows you to specify, in a 1-byte field, the execution diagnostic facility (EDF) status of the program to be autoinstalled. The equated values are:

PGAC_CEDF_YES
EDF can be used with this program.

PGAC_CEDF_NO
EDF cannot be used with this program.

PGAC_DATA_LOCATION
allows you to specify, in a 1-byte field, the data location for task-lifetime storage. The equated values are:

PGAC_LOCATION_BELOW
Task-lifetime storage must be located below 16 MB.

PGAC_LOCATION_ANY
Task-lifetime storage can be below or above 16 MB.

PGAC_EXECUTION_KEY
allows you to specify, in a 1-byte field, the execution key for the program. The equated values are:

PGAC_CICS_KEY
The program is to execute in CICS key.

PGAC_USER_KEY
The program is to execute in user key.

PGAC_LOAD_ATTRIBUTE
allows you to specify, in a 1-byte field, the load attributes for the object. The equated values are:

PGAC_RELOAD
CICS is to load a fresh copy of the object for each request.

PGAC_RESIDENT
CICS is to make the object permanently resident.

PGAC_TRANSIENT
The storage for this object is to be released whenever the use count reaches zero.

PGAC_REUSABLE
CICS can use any copy of the object currently in storage.

PGAC_USE_LPA_COPY
allows you to specify, in a 1-byte field, whether CICS is to use an LPA-resident copy of the program. The equated values are:

PGAC_LPA_YES
CICS is to use a copy from the LPA.

PGAC_LPA_NO
CICS is to load a private copy from its own DFHRPL or dynamic LIBRARY concatenation.

PGAC_EXECUTION_SET
allows you to specify, in a 1-byte field, whether or not the program is restricted to using the distributed program link (DPL) subset of the CICS API. The equated values are:

PGAC_DPLSUBSET
The program is to be restricted to the DPL subset of the EXEC CICS API.

PGAC_FULLAPI
The program can use the full API.

PGAC_REMOTE_SYSID
allows you to specify, in a 4-byte field, the name of the remote system where the program is to execute. CICS function ships any request for this program to the specified remote CICS.

PGAC_REMOTE_PROGID
allows you to specify, in an 8-byte field, the name by which the program is known in the remote CICS region. For a remote program, the remote name defaults to the local name if you set this field to blank.

PGAC_REMOTE_TRANSID
allows you to specify, in a 4-byte field, the name of the CICS mirror transaction under which the program, if remote, is to run. By default, this is set to the name of the CICS mirror transaction, CSMI.

PGAC_DYNAMIC_STATUS
allows you to specify, in a 1-byte field, whether, if the program is the subject of a program-link request, the request can be dynamically routed. The equated values are:

PGAC_DYNAMIC_NO
If the program is the subject of a program-link request, the dynamic routing program is not invoked.

For a distributed program link (DPL) request, the server region on which the program is to execute must be specified explicitly on the REMOTESYSTEM option of the PROGRAM definition or on the SYSID option of the **EXEC CICS LINK** command; otherwise it defaults to the local region.

PGAC_DYNAMIC_YES
If the program is the subject of a program-link request, the dynamic routing program is invoked. Providing that a remote server region is not named explicitly on the SYSID option of the EXEC CICS LINK command, the routing program can route the request to the region on which the program is to execute.

PGAC_CONCURRENCY
allows you to specify, in a 1-byte field, whether or not the program is written to threadsafe standards. The equated values are:

PGAC_QUASIRENT
The program is quasi-reentrant only, and relies on the serialization provided by CICS when accessing shared resources.

The program is restricted to the CICS permitted programming interfaces, and must comply with the CICS quasi-reentrancy rules.

PGAC_THREADSAFE
The program is written to threadsafe standards, and when it accesses shared resources it takes into account the possibility that other programs may be executing concurrently and attempting to modify the same resources.

PGAC_JVM
allows you to specify, in a 1-byte field, whether the program is to be run under a JVM. The equated values are:

PGAC_JVM_YES
The program is a Java bytecode program and must run under the control of a JVM.

PGAC_JVM_NO
The program does not require a JVM for its execution.

PGAC_JVM_CLASS_LEN
allows you to specify, as a two-byte binary value, the length of the Java class name supplied in PGAC_JVM_CLASS_DATA.

PGAC_JVM_CLASS_DATA
allows you to specify, as a 256-byte field, the name of the Java class to be invoked.

PGAC_JVM_PROFID
allows you to specify, in an 8-byte field, the name of the JVM profile to be used for the JVM in which the program is to run.

PGAC_RETURN_CODE
allows you to specify, in a 1-byte field, the autoinstall control program's return code to CICS. The equated values are:

PGAC_RETURN_OK
Install the program definition using the values returned in the communications area parameter list.

PGAC_RETURN_DONT_DEFINE_PROGRAM
Do not define the program.

Sample autoinstall control program for programs, DFHPGADX

The CICS-supplied default autoinstall program is an assembler-language command-level program, named DFHPGADX. The source of the default program is provided in COBOL, PL/I, and C, as well as in assembler language.

The names of the CICS-supplied programs and their associated copy books, and the CICSTS52.CICS libraries in which they can be found, are summarized in Table 31.

Table 31. Sample programs and copy books for program autoinstall

Language	Member name	Library
Executable file:		
Assembler (only)	DFHPGADX	SDFHLOAD
Program source:		
Assembler	DFHPGADX	SDFHSAMP
COBOL	DFHPGAOX	SDFHSAMP
PL/I	DFHPGALX	SDFHSAMP
C	DFHPGAHX	SDFHSAMP
Copy books:		
Assembler	DFHPGACD	SDFHMAC
COBOL	DFHPGACO	SDFHCOB
PL/I	DFHPGACL	SDFHPL1
C	DFHPGACH	SDFHC370

Sample program customization

You can write your autoinstall control program in any of the languages supported by CICS. The control program has full access to the CICS application and system programming interfaces.

If you customize the supplied control program, or write your own version, note these points:

- **Input:** The first two fields of the parameter list are input-only fields and must not be altered by your program.
- **Output:** The remaining fields on the parameter list are input/output or output-only fields, which you can use to specify attributes that override the fields of the model definition.
- Some of the output fields in the parameter list are not applicable to map sets or partition sets. CICS ignores any parameters you specify that are not applicable to the type of object being installed.
- Any attributes you return to CICS in the parameter list are used to modify the model definition, and CICS installs the modified definition. After installation, the definition can be modified normally by using the CICS Explorer **Programs** operations view, the **EXEC CICS SET PROGRAM** command, or the **CEMT SET PROGRAM** command.
- If you modify your control program, you can make the new version available by using the NEWCOPY option or attribute in the view or command.
- You can discard definitions after they have been installed; they are reinstalled when next referenced.
- You must ensure that the parameters you return to CICS are valid, and consistent with other system attributes in your CICS region. For example:
  - Do not return PGAC_LPA_YES on the PGAC_USE_LPA_COPY parameter if CICS is running with the system initialization parameter LPA=NO.
  - Do not return PGAC_USER_KEY (which is the default) on the PGAC_EXECUTION_KEY parameter if the task for which your control program is called is running with CICS-key task-lifetime storage.

    You can determine the storage key for the task by testing the TASKDATAKEY option in its transaction definition with the following **EXEC CICS** commands:

    - **EXEC CICS ADDRESS EIB**
    - **EXEC CICS INQUIRE TRANSACTION(EIBTRANS) TASKDATAKEY(...)**

Important

When you create an autoinstalled program definition, CICS ignores the program language specified on the model program definition. CICS determines the language from the load module itself, when the autoinstalled program is started.

However, CICS does *not* deduce characteristics other than language from the load module. These other program characteristics must be explicitly defined by the autoinstall control program or by RDO. If your programs have varying characteristics (varying AMODE or DATALOCATION requirements, for example), you must be able to distinguish between the various types when using autoinstall. Keep a list of exceptions to the default characteristics, and code your autoinstall control program to reference this list; or you might decide to install explicit RDO definitions of the exceptions.

Resource definition

The autoinstall control program cannot itself be autoinstalled, nor can any program it references; you must create a program resource definition for the control program and for any other programs it references.

You must ensure these definitions are installed in the CICS region during startup by including the group containing the definitions in your startup grouplist. If you specify an invalid name for the control program, CICS disables the program, thus disabling the program autoinstall function.

The following program resource definitions are supplied by CICS for the autoinstall control program; the default is the assembler version, DFHPGADX. If these definitions are not suitable for your use, you can create your own, using RDO or the DFHCSDUP utility.

- Default autoinstall control program definition for DFHPGADX. This defines the assembler version, and its status is set to ENABLED:

```
GROUP(DFHPGAIP)       PROGRAM(DFHPGADX)
DESCRIPTION(Assembler definition for program autoinstall exit)
LANGUAGE(ASSEMBLER)   EXECKEY(CICS)         EXECUTIONSET(FULLAPI)
RELOAD(NO)            RESIDENT(NO)          USAGE(NORMAL)
STATUS(ENABLED)       CEDF(NO)              DATALOCATION(ANY)
CONCURRENCY(THREADSAFE)
```

- Autoinstall control program definition for DFHPGAOX. This defines the CICS-supplied COBOL version, and its status is set to DISABLED:

```
GROUP(DFHPGAIP)       PROGRAM(DFHPGAOX)
DESCRIPTION(COBOL definition for program autoinstall exit)
LANGUAGE(COBOL)       EXECKEY(CICS)         EXECUTIONSET(FULLAPI)
RELOAD(NO)            RESIDENT(NO)          USAGE(NORMAL)
STATUS(DISABLED)      CEDF(NO)              DATALOCATION(ANY)
CONCURRENCY(THREADSAFE)
```

- Autoinstall control program definition for DFHPGAHX. This defines the CICS-supplied C version, and its status is set to DISABLED:

```
GROUP(DFHPGAIP)       PROGRAM(DFHPGAHX)
DESCRIPTION(C definition for program autoinstall exit)
LANGUAGE(C)           EXECKEY(CICS)         EXECUTIONSET(FULLAPI)
RELOAD(NO)            RESIDENT(NO)          USAGE(NORMAL)
STATUS(DISABLED)      CEDF(NO)              DATALOCATION(ANY)
CONCURRENCY(THREADSAFE)
```

- Autoinstall control program definition for DFHPGALX. This defines the CICS-supplied PL/I version, and its status is set to DISABLED:

```
GROUP(DFHPGAIP)       PROGRAM(DFHPGALX)
DESCRIPTION(PL/I definition for program autoinstall exit)
LANGUAGE(PLI)         EXECKEY(CICS)         EXECUTIONSET(FULLAPI)
RELOAD(NO)            RESIDENT(NO)          USAGE(NORMAL)
STATUS(DISABLED)      CEDF(NO)              DATALOCATION(ANY)
CONCURRENCY(THREADSAFE)
```

Testing and debugging your program

You can use the CICS execution diagnostic facility (EDF) to help you test your autoinstall control program. However, EDF is inhibited for programs with names that begin with the letters DFH; so to use EDF you must name your program something other than one of the default names.

About this task

Chapter 19. Writing a dynamic routing program

CICS provides a dynamic routing program that can route transactions initiated from terminals or by a subset of CICS commands, and route program link requests. CICSPlex SM provides a dynamic routing program that can perform workload routing. If these program do not meet your requirements, you can write your own dynamic routing program.

To write a dynamic routing program, you must be familiar with the principles of CICS transaction routing, distributed program links, and dynamic routing. For detailed information about which transactions initiated by START commands, and which program link requests, are eligible for dynamic routing, see Routing transactions invoked by START commands in Getting started.

Restrictions

You cannot use the dynamic routing program to route:

- CICS business transaction services activities and processes.
- Non-terminal-related **EXEC CICS START** requests.
- Inbound web services requests.

To route these types of request you must use the distributed routing program. How to write a distributed routing program is described in Chapter 20, "Writing a distributed routing program," on page 609.

Routing transactions dynamically

Dynamic routing of transactions can be started from user terminals or by eligible terminal-related **EXEC CICS START** commands.

When you define transactions to CICS, you can describe them as *remote* or *local*. Local transactions are always executed in the terminal-owning region; remote transactions can be routed to other regions connected to the terminal-owning region by IPIC, MRO, or APPC (LUTYPE6.2) ISC links. IPIC supports transaction routing of 3270 terminals between CICS TS 4.1 or later regions, where the terminal-owning region (TOR) is uniquely identified by an APPLID.

You can dynamically select both the system to which the transaction is to be routed and the remote name of the transaction, rather than when the transaction is defined to CICS, by using a *dynamic routing program*. The CICS-supplied default routing program is called DFHDYP. Its source-level code is supplied in assembler language, COBOL, PL/I, and C versions. You can write your own program in any of these languages, using the default program as a model.

Dynamic transactions

When you want to route transactions dynamically, you must define them with the value DYNAMIC(YES) and supply values for both the remote and the local options.

Defining the transactions in this way allows CICS to select the appropriate values when the transaction is routed, and to ignore those values that are not needed.

For information about defining transactions for dynamic transaction routing, see Defining transactions for transaction routing, in the *CICS Intercommunication Guide*.

When the dynamic routing program is invoked

For transactions initiated from user terminals or by eligible terminal-related **EXEC CICS START** commands, CICS calls the dynamic routing program as follows:

- When a transaction defined as DYNAMIC(YES) is initiated.

  Note:

  1. If a transaction definition is not found, CICS uses the common transaction definition specified on the DTRTRAN system initialization parameter.
  2. If a transaction defined as DYNAMIC(YES) and initiated by a terminal-related **EXEC CICS START** command is ineligible for dynamic routing, the routing program is invoked for notification only—it cannot route the transaction.
- If an error occurs in route selection—for example, if the target region returned by the routing program on its initial (route selection) call is unavailable. This gives the routing program an opportunity to specify an alternate target. This process iterates until the routing program selects a target that is available or sets a nonzero return code.
- After the routed transaction has completed, if the routing program has requested to be reinvoked at termination.
- If the routed transaction abends, if the routing program has requested to be reinvoked at termination.

Figure 67 shows the points at which the dynamic routing program is invoked.

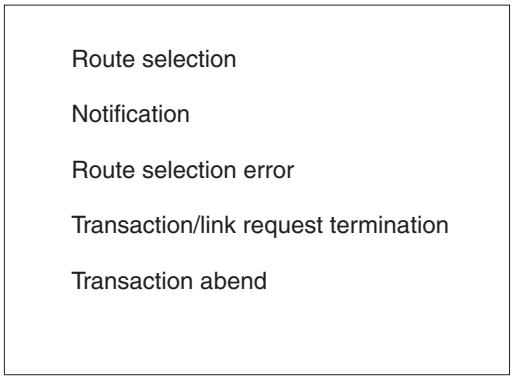


Figure 67. When the dynamic routing program is invoked

Information passed to the dynamic routing program

The CICS relay program, DFHAPRT, passes information to the dynamic routing program by using a communications area. The communications area contains fields that are mapped by the DSECT DFHDYPDS.

The DFHDYPDS DSECT is described in detail in "Parameters passed to the dynamic routing program" on page 593. For transaction routing, here is the data that is passed to the dynamic routing program in the communications area:

- The SYSID of the remote CICS region that was specified when the transaction was installed
- The netname of the remote CICS region

- The name of the remote transaction
- The priority of the relay transaction task, for MRO and IPIC connections only
- Whether the request is to be queued if no sessions are immediately available to the remote CICS region
- The address of the communications area of the remote transaction
- The address of a copy of the terminal input/output area (TIOA)of the transaction
- A task-local user data area.
- Application context data.

The communications area DSECT contains comments to describe the information that is passed.

The dynamic routing program can accept these values, or change them, or tell CICS not to continue routing the transaction. The values that are used depend on the function that is being performed; that is, some values might be ignored.

The information that is passed to the dynamic routing program indicates whether the transaction is being routed dynamically or statically. If the transaction is being routed dynamically, the dynamic routing program can change the SYSID or netname to determine where the transaction is to run.

Sometimes, the dynamic routing program is invoked for transactions that are routed statically. It is invoked if a transaction defined as DYNAMIC(YES) is initiated by automatic transaction initiation (ATI), for example, by the expiry of an interval control start request, but the transaction is ineligible for dynamic routing. In this case, the dynamic routing program is called only to notify itself of where the transaction is going to run. It cannot change the remote system name, and any changes it makes to the SYSID or NETNAME fields in the communications area are ignored.

For transactions that are run remotely, either because they are defined as remote or because they are dynamically routed to a remote CICS region, CICS monitoring is informed of the SYSID of the remote CICS region. For transactions that the dynamic routing program routes locally, the monitoring field is set to nulls.

Changing the target CICS region

The dynamic routing program can change the target CICS region by modifying the system identifier (sysid) and netname of the default CICS region to which the transaction is to be routed.

The communications area passed to the dynamic routing program initially contains the system identifier (sysid) and netname of the default CICS region to which the transaction is to be routed. These are derived from the value of the REMOTESYSTEM option of the installed transaction definition. If the transaction definition does not specify a REMOTESYSTEM value, the sysid and netname passed are those of the local CICS region.

The dynamic routing program can change the sysid and netname. If it does so when it is invoked for route selection, the region to which the transaction is routed is determined as follows:

- The NETNAME and the SYSID are not changed.

CICS tries to route to the SYSID as originally specified in the communications area.

- The NETNAME is not changed, but the SYSID is changed.

  CICS updates the communications area with the NETNAME corresponding to the new SYSID, and tries to route to the new SYSID.
- The NETNAME is changed, but the SYSID is not changed.

  CICS updates the communications area with a SYSID corresponding to the new NETNAME, and tries to route to the new SYSID.
- The NETNAME is changed **and** the SYSID is changed.

  CICS overwrites the communications area with a SYSID corresponding to the new NETNAME, and tries to route to that new SYSID.

If the NETNAME specified is invalid, or cannot be found, SYSIDERR is returned to the dynamic routing program—which may deal with the error by returning a different SYSID or NETNAME—see "If the system is unavailable or unknown" on page 579.

If the routing program changes the SYSID or NETNAME when it is invoked for notification, the changes have no effect.

Using a common transaction definition in the TOR

It is good practice to use a single, common definition for all remote transactions that are to be dynamically routed.

The name of the common definition is specified on the DTRTRAN system initialization parameter. You can use the REMOTESYSTEM option of the common definition to specify a default AOR to which transactions are to be routed. For information about defining remote transactions for dynamic transaction routing, see Defining remote resources, in the *CICS Intercommunication Guide*.

Important: To route a transaction defined by the DTRTRAN definition, your dynamic routing program must set the DYRDTRRJ field of the communications area to 'N' (the default is 'Y'). If you leave DYTDTRRJ set to 'Y', the transaction is rejected.

You can test the DYRDTRXN field to check if the transaction passed to your routing program is defined by the DTRTRAN definition. Figure 68 contains skeleton code for routing transactions defined by DTRTRAN.

```
if DYRDTRXN='Y' then          /* Is DYP invoked because of DTRTRAN */
do                            /* .. Yes                            */
  Call Find_AOR(sysid)        /*    Select the SYSID of the AOR    */
  if rc=0 then                /*    Is AOR available?              */
  do                          /*    .. Yes                         */
    DYRRETC=RETCOD0           /*       Set OK Return Code          */
    DYRSYSID=sysid            /*       Set the sysid               */
    DYRDTRRJ='N'              /*       Don't reject DTRTRAN defns  */
    ...                       /*       Set other commarea fields   */
  end                         /*                                   */
  else                        /*    .. No                          */
    ...                       /*       AOR unavailable logic       */
end                           /*                                   */
```

Figure 68. Example pseudocode to route transactions defined by DTRTRAN

Changing the program name

For transactions defined as DYNAMIC, on invocation of the routing program the DYRLPROG field in the communications area contains the name of the initial program associated with the transaction to be routed. If you decide to route the transaction locally, you can use this field to specify an alternative program to be run.

For example, if all remote CICS regions are unavailable and the transaction cannot be routed, you may want to run a program in the local CICS terminal-owning region to send an appropriate message to the user.

Telling CICS whether to route or terminate a transaction

When the routing program is invoked for routing, it can choose whether the transaction should be routed or terminated.

If you want the transaction to be routed, whether you have changed any values or not, return a zero value to CICS in field DYRRETC of the communications area. When you return control to CICS with return code zero, CICS first compares the returned SYSID with its own local SYSID:

- If the SYSIDs are the same (or the returned SYSID is blank) CICS executes the transaction locally.
- If the two SYSIDs are not the same, CICS routes the transaction to the remote CICS region, using the remote transaction name.

If you want to terminate the transaction with a message or an abend, set a return code of X'8' (or any other non-zero return code other than X'4').

If you want to terminate the transaction without issuing a message or abend, set a return code of X'4'.

Warning: Setting a return code of X'4' for APPC transaction routing leads to unpredictable results, and should be avoided.

Returning a value in DYRRETC has no effect when the routing program is invoked for notification or at termination of the transaction.

If the system is unavailable or unknown

The dynamic routing program is invoked again if the remote system name that you specify on the route selection call is not known or is unavailable.

When the program is re-invoked, you have a choice of actions:

- You can tell CICS not to continue trying to route the transaction, by issuing a return code of '8' in DYRRETC. If the reason for the error is that the system is unavailable, CICS issues message 'DFHAC2014' or 'DFHAC2029' to the terminal user. If the reason for the error is that the system is unknown, DFHAPRT abends the transaction.
- You can tell CICS to terminate the transaction without issuing a message or abend by placing a return code of '4' in DYRRETC. However, note the warning about setting return code '4'.
- If the reason for the error is that no sessions are immediately available to the remote system, you can reset field DYRQUEUE to 'Y' (it must previously have been set to 'N'—the request is **not** to be queued—for this error to occur), issue a return code of '0' in DYRRETC, and try to route the transaction again.

If you try to route the transaction again **without** resetting DYRQUEUE to 'Y' (and without changing the sysid), and the system is still unavailable, DFHDYP is reinvoked. If you then choose to set return code '8', CICS terminates the transaction with message 'DFHAC2030'.

- You can change the sysid, and issue a return code of '0' in DYRRETC to try to route the transaction again. Note that if you change the sysid, you may also need to supply a different remote transaction ID. You need to do this if, for example, the transaction has a different remote transaction name on each system.

A count of the times the routing program has been invoked for routing purposes for this transaction is passed in field DYRCOUNT. Use this count to help you decide when to stop trying to route the transaction.

Invoking the dynamic routing program at end of routed transactions

If you want your dynamic routing program to be invoked again when the routed transaction has completed, you must set the DYROPTER field in the communications area to 'Y' before returning control to CICS.

You might want to do this, for example, if you are keeping a count of the number of transactions currently executing on a particular CICS region. However, during this reinvocation, the dynamic routing program should update only its own resources. This is because, at this stage, the final command to the terminal from the application program in the AOR may be pending, and the dynamic routing program is about to terminate.

Invoking the dynamic routing program on abend

If you have set DYROPTER to 'Y', and the routed transaction abends, the dynamic routing program is invoked again to notify it of the abend. You could use this invocation to initiate a user-defined program in response to the transaction abend.

If the routed transaction abends, the APRT program in the TOR:

1. Passes back a response to the CICS transaction manager indicating that a transaction abend has occurred
2. If the dynamic routing program requested to be reinvoked at termination of the transaction (by setting DYROPTER to 'Y' when invoked for routing), reinvokes the dynamic routing program
3. Returns to CICS transaction manager.

Modifying the initial terminal data

The dynamic routing program must not perform an **EXEC CICS RECEIVE** or an **EXEC CICS GDS RECEIVE** command, because this prevents the routed-to transaction from obtaining the initial terminal data.

The CICS relay program, DFHAPRT, places a *copy* of the user's initial terminal input into a separate buffer. This information includes SNA presentation services headers for APPC mapped and unmapped conversations. A pointer to this buffer (DYRBPNTR), and its length (DYRBLGTH), are provided in the communications area passed from DFHAPRT to the dynamic routing program.

Note that:

- The buffer pointed to by DYRBPNTR contains the data that arrived in the first request unit (RU) of the message. If the RU size is large enough to hold the full

message, the buffer contains the full message. However, if the RU size is less than the message length, the buffer contains only the data from the first RU (even if the buffer itself is large enough to hold the full message).

- The length field DYRBLGTH is the length of the *message*, not the length of the data in the buffer. DYRBLGTH contains the length of data in the buffer only if the full message arrived in a single RU.
- If all the following are true, no initial terminal input data is passed to the routing program:
  1. The routing program is running in the AOR.
  2. The original request was transaction-routed from the TOR.
  3. The originating facility is an APPC parallel session.

Because the transaction profile has not been queried at this point, uppercase translation has not been performed on the input data unless UCTRAN(YES) is specified on the TYPETERM definition.

Sometimes you may want to modify the initial data input by the user. (It may be necessary to do this if, for example, you change the ID of the remote transaction, using field DYRTRAN of the communications area.) To modify the input data, your routing program should, when invoked for route selection:

1. Copy the input data pointed to by DYRBPNTR into a named variable, of length DYRBLGTH
2. Modify the data in the named variable
3. Use the INPUTMSG option of the EXEC CICS RETURN command to make the modified data available to the application program.

For guidance information about using INPUTMSG on EXEC CICS RETURN commands, see the other methods described in INPUTMSG, in the *CICS Application Programming Guide*. For programming information about the INPUTMSG option, see RETURN, in the *CICS Application Programming Reference* manual.

Note: If, after modifying the input data, the dynamic routing program is reinvoked because an error occurs in routing to the selected transaction, it should "remember" that it has modified the original user-input.

Modifying the application's communications area

Sometimes you want to modify the routed application's communications area. For example, if your routing program changes the ID of the remote transaction, it may also need to change the input communications area passed to the routed application.

Field DYRACMAA of the routing program's communications area enables you to do this; it is a pointer to the application's communications area.

See also "Modifying the application's containers" on page 592.

Receiving information from a routed transaction

If your dynamic routing program chooses to be reinvoked at the end of a routed transaction, it can obtain information about the transaction by monitoring its output communications area and output TIOA.

Monitoring the output communications area

A routed transaction can pass information back to the dynamic transaction routing program in its output communications area. When invoked at transaction termination, your routing program can examine the output communications area (pointed to by DYRACMAA).

This is an example of how this facility could be used:

- You have a CICSplex consisting of sets of functionally-equivalent TORs and AORs, and need to identify any inter-transaction affinities that may affect transaction routing. You could use the CICS Interdependency Analyzer to do this, but there are some affinities that the utility cannot detect (for example, those created by non-CICS functions). Also, some transactions may sometimes create affinities, and sometimes not.

  For information about the CICS Interdependency Analyzer, see CICS Interdependency Analyzer.

  However, the routed transactions themselves know when an affinity is created, and can communicate this to the dynamic transaction routing program. The routing program is then able to route such transactions accordingly.

See also “Modifying the application’s containers” on page 592.

Monitoring the output TIOA

When invoked at transaction termination, your routing program can examine the copy of the routed transaction's output TIOA pointed to by DYRBPNTR.

This can be useful, for example, to guard against the situation where one AOR in a CICSplex develops software problems. These may be reported by means of a message to the end user, rather than by a transaction abend. If this happens, the routing program is unaware of the failure and cannot bypass the AOR that has the problem. By reading the output TIOA, your routing program can check for messages indicating specific kinds of failure, and bypass any AOR that is affected.

Some processing considerations

- Any of the EXEC CICS commands (except EXEC CICS RECEIVE—see “Modifying the initial terminal data” on page 580) can be issued from the routing program. You are likely to find the EXEC CICS INQUIRE CONNECTION and INQUIRE IRC commands particularly useful if you want to confirm that a link is available before routing a transaction. The EXEC CICS INQUIRE and SET commands are described in System commands, in the *CICS System Programming Reference* manual.
- Although the routing program can issue any EXEC CICS command, you should consider carefully the effect of commands that alter protected resources, because changes to those resources may be committed or backed out inadvertently as a result of logic in the routed transaction. You should also consider carefully the effect of EXEC CICS SYNCPOINT and ABEND commands on APPC transaction routing.
- If you want to keep information about how transactions are routed, it must be done in the user routing program, perhaps by writing the information to a temporary storage queue associated with this terminal.
- Several transactions can form a single conversation with the end user. At the start of the conversation, resources are allocated to record the state of the conversation. Because these resources are local to the system to which the first transaction in the conversation was routed, the routing program must be able to continue to route to this system until the end of the conversation.

- It is important to avoid creating “tangled daisychains”: for any transaction that is being dynamically routed, you must avoid routing back to a node that has previously been routed from.
- The dynamic routing program can be RMODE ANY but must be AMODE 31.

Unit of work considerations

Depending on the terminal type, the CICS relay program, the dynamic routing program, and the routed transaction may constitute a single unit of work. Any protected resources owned by the dynamic routing program could therefore be affected by the syncpoint activity of the routed transaction. This means that these resources may be committed or backed out inadvertently by the routed transaction. If you want to avoid this, you have to define the routing program's resources as unprotected rather than protected.

Routing DPL requests dynamically

For a program-link request to be eligible for dynamic routing, the remote program must either be defined to the local system as DYNAMIC(YES) or not be defined to the local system.

Note: If the program specified on an **EXEC CICS LINK** command without a SYSID is not currently defined, what happens next depends on whether program autoinstall is active:

- If program autoinstall is inactive, the dynamic routing program is invoked.
- If program autoinstall is active, the autoinstall user program is invoked. The dynamic routing program is then invoked only if the autoinstall user program:
  - Installs a program definition that specifies DYNAMIC(YES), or
  - Does not install a program definition.

  See “Autoinstall programs started by **EXEC CICS LINK** commands” on page 566.

As well as CICS-to-CICS DPL calls instigated by **EXEC CICS LINK PROGRAM** commands, program-link requests received from outside CICS can also be dynamically routed. For example, all the following types of program-link request can be dynamically routed:

- Calls from external CICS interface (EXCI) client programs
- External Call Interface (ECI) calls from any of the CICS Client workstation products
- Distributed Computing Environment (DCE) remote procedure calls (RPCs)
- ONC/RPC calls.

A program-link request received from outside CICS can be dynamically routed by:

- Defining the program to CICS TS as DYNAMIC(YES)
- Coding your dynamic routing program to route the request.

When the dynamic routing program is invoked

CICS can invoke the dynamic routing program for eligible program-link requests.

CICS invokes the dynamic routing program in the following circumstances:

- Before the linked-to program is executed, to either:
  - Obtain the SYSID of the region to which the link should be routed.

Note: The address of the caller's communications area (COMMAREA) is passed to the routing program, which can therefore route requests by COMMAREA contents if this is appropriate.

– Notify the routing program of a statically-routed request. This occurs if the program is defined as DYNAMIC(YES)—or is not defined—but the caller specifies the name of a remote region on the SYSID option of the LINK command.

  In this case, specifying the target region explicitly takes precedence over any SYSID returned by the dynamic routing program.

- If an error occurs in route selection—for example, if the SYSID returned by the dynamic routing program is unavailable or unknown, or the link fails on the specified target region—to provide an alternate SYSID. This process iterates until either the program-link is successful or the return code from the dynamic routing program is not equal to zero.

  Special case: care!

  If all the following are true, the route selection call fails *but the routing program is not reinvoked for a route selection error*:

  1. The program is not defined on the local region.
  2. Program autoinstall is not active on the local region.
  3. On the route selection call, the routing program routes the link request to the local region.

  Therefore, to dynamically route a program-link request *that the routing program may route locally*, you should do either of the following:

  1. Install a program definition on the local region, specifying DYNAMIC(YES).
  2. Set program autoinstall active, using it to install a definition that specifies DYNAMIC(YES).

- After the link request has completed, if reinvocation was requested by the routing program.
- If an abend is detected after the link request has been shipped to the specified remote system, if reinvocation was requested by the routing program.
- At the end of a unit of work, to issue a notification that the unit of work is complete, if reinvocation was requested by the routing program. CICSPlex SM workload management uses these notifications to manage UOW affinities.

Figure 67 on page 576 shows the points at which the dynamic routing program is invoked.

Changing the target CICS region

The communications area passed to the dynamic routing program initially contains the system identifier (sysid) and netname of the default CICS region to which the link request is to be routed. These are derived from the value of the REMOTESYSTEM option of the installed program definition. If REMOTESYSTEM is not specified, or there is no program definition, the sysid and netname passed are those of the local CICS region.

The dynamic routing program can change the sysid and netname. If it does so when it is invoked for route selection, the region to which the link request is routed is determined as follows:

- The NETNAME and the SYSID are not changed.

CICS tries to route to the SYSID as originally specified in the communications area.

- The NETNAME is not changed, but the SYSID is changed.

  CICS updates the communications area with the NETNAME corresponding to the new SYSID, and tries to route the request to the new SYSID.
- The NETNAME is changed, but the SYSID is not changed.

  CICS updates the communications area with a SYSID corresponding to the new NETNAME, and tries to route the request to the new SYSID.
- The NETNAME is changed **and** the SYSID is changed.

  CICS overwrites the communications area with a SYSID corresponding to the new NETNAME, and tries to route the request to that new SYSID.

If the REMOTESYSTEM option of the program definition names a remote region, the routing program cannot route the request locally.

You can route DPL requests over both IPIC and ISC over SNA connections. If there is both an IPIC connection and an ISC over SNA connection to the selected target region, and both are named the same, the IPIC connection takes precedence. That is, if remote SYSID "CICB" is defined by both an IPCONN definition and a CONNECTION definition, CICS uses the IPCONN connection.

If the NETNAME specified is invalid, or cannot be found, SYSIDERR is returned to the dynamic routing program—which may deal with the error by returning a different SYSID or NETNAME—see "If an error occurs in route selection" on page 586.

If the routing program changes the SYSID or NETNAME when it is invoked for notification, the changes have no effect.

Changing the program name

When the routing program is invoked for route selection or for notification of a program-link request, the DYRLPROG field in the communications area contains the name of the program to be linked, obtained using the following sequence:

1. From the REMOTENAME option of the installed program definition
2. If REMOTENAME is not specified, or there is no program definition, from the PROGRAM option of the EXEC CICS LINK command.

When it is invoked for routing [2] (not for notification of a statically-routed request), your routing program can, by overwriting the DYRLPROG field, specify that an alternative program is to be linked. You can specify a local or remote program, depending on the region to which the request is to be routed.

Changing the transaction ID

When it is invoked for routing (not for notification of a statically-routed request), your routing program can change the remote transaction ID by overwriting the DYRTRAN field in the communications area.

A transaction identifier is always associated with each dynamic program-link request. CICS obtains the transaction ID using the following sequence:

1. From the TRANSID option on the LINK command

2. By "invoked for routing" we mean both "invoked for route selection" and "invoked because an error occurred in route selection".

2. From the TRANSID option on the program definition
3. 'CSMI', the generic mirror transaction. This is the default if neither of the TRANSID options are specified.

Note: If you use CICSPlex System Manager to route your program-link requests, the transaction ID becomes highly significant, because CICSPlex System Manager's routing logic is transaction-based. CICSPlex System Manager routes each DPL request according to the rules specified for its associated transaction.

The CICSPlex System Manager system programmer can use the EYU9WRAM user-replaceable module to change the transaction ID associated with a DPL request.

Telling CICS whether to route or terminate a DPL request

When the routing program is invoked for routing, it can choose whether the link request should be routed or rejected. If you want the request to be routed, whether you have changed any values or not, return a zero value to CICS in field DYRRETC of the communications area.

When you return control to CICS with return code zero, CICS first compares the returned SYSID with its own local SYSID:

- If the SYSIDs are the same (or the returned SYSID is blank) CICS executes the link request locally.
- If the two SYSIDs are not the same, CICS routes the request to the remote CICS region, using the returned program and transaction names.

To make CICS reject the link request, return a non-zero value. The program that issued the EXEC CICS LINK command receives a PGMIDERR condition, with a RESP2 value of 25.

Returning a value in DYRRETC has no effect when the routing program is invoked for notification or at termination of the request.

If an error occurs in route selection

If an error occurs in route selection—for example, if the SYSID returned by the dynamic routing program is unavailable or unknown, or the link fails on the specified target region— the dynamic routing program is invoked again.

When the program is re-invoked, you have a choice of actions:

- You can tell CICS not to continue trying to route the request, by issuing a non-zero return code in DYRRETC.
- If the reason for the error is that no sessions are immediately available to the remote system, you can reset field DYRQUEUE to 'Y' (it must previously have been set to 'N'—the request is **not** to be queued—for this error to occur), issue a return code of '0' in DYRRETC, and try to route the request again.
- You can change the sysid, and issue a return code of '0' in DYRRETC to try to route the request again. Note that if you change the sysid, you may also need to supply a different remote program name or transaction ID.

A count of the times the routing program has been invoked for routing purposes for this request is passed in field DYRCOUNT. Use this count to help you decide when to stop trying to route the transaction.

Special case—care!

If all the following are true, the route selection call fails *but the routing program is not reinvoked for a route selection error*:

1. The program is not defined on the local region.
2. Program autoinstall is not active on the local region.
3. On the route selection call, the routing program routes the link request to the local region.

Therefore, to dynamically route a program-link request *that the routing program may route locally*, you should do either of the following:

1. Install a program definition on the local system, specifying DYNAMIC(YES).
2. Set program autoinstall active, using it to install a definition that specifies DYNAMIC(YES).

Using the XPCERES exit to check the availability of resources on the target region

You can use an XPCERES global user exit program to check that all resources required by the linked-to program are available on the target region.

The XPCERES exit is invoked, if enabled, on the target region before CICS processes a dynamically-routed program-link request.

If, for example, the linked-to program is disabled on the target region, or a required file is missing, your exit program can give the dynamic routing program the opportunity to route the request to a different region. To do this, it should set a return code of UERCRESU. This causes CICS to:

1. Return a RESUNAVAIL condition on the EXEC CICS LINK command executed by the mirror on the target region. (This condition is *not* returned to the application program.)
2. Set the DYRERROR field of the routing program's communications area to 'F'—resource unavailable.
3. Reinvoke the routing program, on the routing region, for route selection failure—see “If an error occurs in route selection” on page 586.

For information about writing an XPCERES global user exit program, see The XPCERES global user exit.

If a required resource is unavailable on the target region, but the XPCERES exit is unavailable or disabled (or is enabled but does not set the UERCRESU return code), the client program receives an error response.

Invoking the dynamic routing program at end of routed requests

If you want your dynamic routing program to be invoked again when the routed request has completed, you must set the DYROPTER field in the communications area to 'Y' before returning control to CICS.

You might want to do this, for example, if you are keeping a count of the number of link requests currently executing on a particular CICS region.

If you have set DYROPTER to 'Y', and the linked program abends, the dynamic routing program is invoked to notify it of the abend.

Modifying the application's input communications area

Sometimes you may want to modify the routed application's communications area. For example, if your routing program changes the name of the remote program, it may also need to change the input communications area passed to the program.

Field DYRACMAA of the routing program's communications area enables you to do this; it is a pointer to the application's communications area (or null, if no communications area was specified on the LINK command).

See also “Modifying the application’s containers” on page 592.

Monitoring the application's output communications area

A routed application can pass information back to the dynamic transaction routing program in its output communications area. If your dynamic routing program chooses to be reinvoked at the end of a routed DPL request, it can examine the output communications area (if any) pointed to by DYRACMAA.

See also “Modifying the application’s containers” on page 592.

Some processing considerations

A dynamic routing program has the following processing considerations.

- When invoked for program-link requests, the dynamic routing program should restrict its use of EXEC CICS commands to those in the DPL subset. For information about which commands constitute the DPL subset, see Exception conditions for LINK command in the *CICS Application Programming Reference*.
- Although the routing program can issue any EXEC CICS command in the DPL subset, consider carefully the effect of commands that alter protected resources, because changes to those resources might be committed or backed out inadvertently as a result of logic in the routed program.
- If you want to keep information about how link requests are routed, this must be done in the user routing program, perhaps by writing the information to a temporary storage queue.
- Avoid creating “tangled daisychains”. For any program-link request that is being dynamically routed, avoid routing back to a node that has previously been routed from. For more details, see Daisy-chaining of DPL requests in the *CICS Intercommunication Guide*.
- The dynamic routing program can be RMODE ANY, but must be AMODE 31.

Unit of work considerations

The client program, the dynamic routing program, and possibly the server program constitute a single unit of work. Any recoverable resources owned by the dynamic routing program could therefore be affected by the syncpoint activity of the client program. This means that these resources may be committed or backed out inadvertently by the client program. If you want to avoid this, you have to define the routing program's resources as non-recoverable.

For information about the syncpoint activity of DPL client and server programs, see The server program, in the *CICS Intercommunication Guide*.

Routing bridge requests dynamically

To run a 3270 user transaction under the control of the bridge, a client program must first issue a LINK, ECI or EXCI call to DFHL3270 running in the bridge router region, passing a COMMAREA that contains the bridge inbound message header (BRIH).

The BRIH contains the name of the target user transaction. DFHL3270 (the bridge program) then links to the CICS driver program, passing the COMMAREA. If the user transaction is eligible for dynamic routing, DFHL3270 calls the dynamic routing program to determine the target system where the driver program will execute.

The user transaction always executes in the same region as the driver program. The client request to run the user transaction is dynamically routed, not the user transaction.

The resource definition of the target transaction on the router region is used to determine if the bridge request to the driver program is eligible for dynamic routing. If the target user transaction is not defined in the router region, the common transaction definition specified on the DTRTRAN system initialization parameter is used to determine if the request is eligible for dynamic routing.

In session mode, the target system of the first user transaction request determines where all subsequent user transaction requests in the session are routed. Remote requests can be routed to other regions connected to the router region by MRO links, or to other systems that are connected by APPC (LUTYPE6.2) ISC links.

Note: The local system is the CICS router region where the dynamic routing program is executing.

The dynamic routing program is invoked in the following cases:

- In single transaction mode when the transaction is defined as DYNAMIC(YES), or the transaction is not defined and the DTRTRAN transaction is defined as DYNAMIC(YES).
- In session mode when the first user transaction is defined as DYNAMIC(YES), or the transaction is not defined and the DTRTRAN transaction is defined as DYNAMIC(YES).
- In session mode when subsequent user transactions are defined as DYNAMIC(YES), or the transaction is not defined and the DTRTRAN transaction is defined as DYNAMIC(YES). In this case, the target system has already been determined by the first user transaction of the session, so the routing program is only invoked for notification; it cannot change the target system of the request.
- If an error occurs in route selection, for example, if the target region returned by the routing program on its initial (route selection) call is unavailable. This allows the routing program to specify an alternate target. This process iterates until the routing program selects a target that is available, or sets a non-zero return code.
- After the user transaction has completed, if the routing program has requested to be reinvoked at termination.

Changing bridge request parameters

The communications area passed to the dynamic routing program initially contains parameters and pointers some of which you can change.

The communications area passed to the dynamic routing program initially contains parameters and pointers that you can examine. These are all described in "Parameters passed to the dynamic routing program" on page 593. The only parameters that you can change for a Link3270 bridge request are:

- The system identifier (SYSID) and netname of the CICS region to which the request is to be routed
- The transaction identifier (TRANSID) of the target user application that is to be run under control of the Link3270 bridge
- The dispatcher priority of the user transaction in the AOR
- A task-local user data area
- DTRTRAN indicators
- Termination option

Changing the Link3270 bridge request SYSID

The initial values of the SYSID and netname of the default CICS region to which the request is to be routed are derived from the value of the REMOTESYSTEM option of the installed user program definition. If REMOTESYSTEM is not specified, or there is no program definition, the sysid and netname passed are those of the local CICS region.

The region to which the request is routed is determined as follows:

- The NETNAME and the SYSID are not changed.

  CICS tries to route to the SYSID as originally specified in the communications area.
- The NETNAME is not changed, but the SYSID is changed.

  CICS updates the communications area with the NETNAME corresponding to the new SYSID, and tries to route the request to the new SYSID.
- The NETNAME is changed, but the SYSID is not changed.

  CICS updates the communications area with a SYSID corresponding to the new NETNAME, and tries to route the request to the new SYSID.
- The NETNAME is changed **and** the SYSID is changed.

  CICS overwrites the communications area with a SYSID corresponding to the new NETNAME, and tries to route the request to that new SYSID.

If the NETNAME specified is invalid, or cannot be found, SYSIDERR is returned to the dynamic routing program—which may deal with the error by returning a different SYSID or NETNAME—see "Handling route selection errors of Link3270 bridge requests" on page 591.

When you return control to CICS with return code zero, CICS first compares the returned SYSID with its own local SYSID:

- If the SYSIDs are the same (or the returned SYSID is blank) CICS executes the link request locally.
- If the two SYSIDs are not the same, CICS routes the request to the remote CICS region, using the returned transaction name.

Changing the bridge request TRANSID

The TRANSID of the target user transaction is passed to the dynamic routing program in DYRTRAN. You can change this by overwriting the DYRTRAN field in the communications area.

Changing the Link3270 bridge request transaction priority

You can change the dispatching priority of the user transaction by specifying the priority in DYRPRTY and putting "Y" in DYRRTPRI. This priority will override the priority specified in the TRANSACTION resource definition in the AOR.

Rejecting a Link3270 bridge request

When the routing program is invoked for routing, it can choose whether the link request should be routed or rejected. If you want the request to be routed, whether you have changed any values or not, return a zero value to CICS in field DYRRETC of the communications area.

The routing program can reject the request by returning a value of 4 or 8 in field DYRRETC.

The BRIH returned to the client contains a return code value indicating that the routing program has rejected the request. The BRIH compcode gives further information about the last attempt to route the request by the routing program. If the routing program placed a return code value of 8 into field DYRRETC a message is issued with the details of the last attempt to route the request.

Returning a value in DYRRETC has no effect when the routing program is invoked at request termination or when a notify call is being made.

Handling route selection errors of Link3270 bridge requests

If an error occurs in route selection—for example, if the SYSID returned by the dynamic routing program is unavailable or unknown, or the link fails on the specified target region—the dynamic routing program is invoked again.

When this happens, you have a choice of actions:

- You can tell CICS not to continue trying to route the request, by issuing a non-zero return code in DYRRETC.
- You can change the sysid, and issue a return code of '0' in DYRRETC to try to route the request again. Note that if you change the sysid, you may also need to supply a different transaction ID.

A count of the times the routing program has been invoked for routing purposes for this request is passed in field DYRCOUNT. Use this count to help you decide when to stop trying to route the transaction.

Using the XPCERES exit to check the availability of resources on the target region

You can use an XPCERES global user exit program to check that all resources required by the 3270 user transaction are available on the target region.

The exit is invoked, if enabled, on the target region before CICS processes a dynamically-routed Link3270 bridge request.

If, for example, the 3270 user transaction is disabled on the target region, or a required file is missing, your exit program can give the dynamic routing program the opportunity to route the request to a different region. To do this, it should set a return code of UERCRESU. This causes CICS to:

1. Return a RESUNAVAIL condition on the EXEC CICS LINK call to DFHL3270 executed by the mirror on the target region

2. Set the DYRERROR field of the routing program's communications area to 'F'—resource unavailable
3. Reinvoke the routing program, on the routing region, for route selection failure—see “Handling route selection errors of Link3270 bridge requests” on page 591

For information about writing an XPCERES global user exit program, see The XPCERES global user exit.

If a required resource is unavailable on the target region, but the XPCERES exit is unavailable or disabled (or is enabled but does not set the UERCRESU return code), the client program receives an error response.

Re-invoking the dynamic routing program after Link3270 bridge requests

If you want your dynamic routing program to be invoked again when the routed request has completed, you must set the DYROPTER field in the communications area to 'Y' before returning control to CICS.

You might want to do this, for example, if you are keeping a count of the number of link requests currently executing on a particular CICS region.

If you have set DYROPTER to 'Y', and the linked program abends, the dynamic routing program is invoked to notify it of the abend.

Link3270 bridge dynamic routing considerations

A dynamic routing program has the following Link3270 bridge considerations.

- If you use the DTRTRAN definition to route the Link3270 request, the routing program must set the DYRTTRRJ field of the communication area to N (the default is Y). If you leave DTRDTRRJ set to Y, the request will be rejected. You can test the DYRDTRXN field to check whether the transaction passed to your routing program is defined by the DTRTRAN definition.
- When invoked for Link3270 bridge requests, the dynamic routing program should restrict its use of EXEC CICS commands to those in the DPL subset. For information about which commands constitute the DPL subset, see in the *CICS Application Programming Reference*.
- Although the routing program can issue any EXEC CICS command in the DPL subset, consider carefully the effect of commands that alter protected resources, because changes to those resources might be committed or backed out inadvertently as a result of logic in the routed program.
- If you want to keep information about how link requests are routed, this must be done in the user routing program, perhaps by writing the information to a temporary storage queue.
- The dynamic routing program can be RMODE ANY, but must be AMODE 31.

Modifying the application’s containers

This section applies to the routing of:

- Transactions started by terminal-related START requests (described in “Routing transactions dynamically” on page 575)
- Program-link (DPL) requests (described in “Routing DPL requests dynamically” on page 583)

- Non-terminal-related START requests (described in "Routing non-terminal-related START requests" on page 614)

If the user application uses a channel, rather than a communications area, the routing program is given, in field DYRCHANL, the name of the channel. Because the routing program is given the *name* of the channel, not its address, it is unable to use the contents of DYRCHANL to inspect or change the contents of the channel's containers.

However, an application that uses a channel can create, within the channel, a special container named DFHROUTE. If the application issues a LINK or terminal-related START request (but not a non-terminal-related START request) that is to be dynamically routed, the dynamic routing program is given, in the DYRACMAA field of DFHDYPDS, the address of the DFHROUTE container, and can inspect and change its contents.

Routing by user ID

Optionally, your routing program can route requests based on the CICS user ID (userid) associated with the request. The DYRUSERID field of the communications area contains the user ID. When it is invoked for routing or because of a route-selection error, your routing program can base its routing decision on the contents of this field.

For details of how the userid is set for different types of request, see the description of the DYRUSERID field in "Parameters passed to the dynamic routing program."

Parameters passed to the dynamic routing program

Parameters are passed from the CICS relay program to the dynamic routing program by a communications area (COMMAREA) or container. The copybook DFHDYPDS maps the COMMAREA or container, which is in the appropriate CICS library for all the supported programming languages.

The same information is passed to both the dynamic routing program and the distributed routing program. Some parameters are meaningful to one routing program but not to the other. Some parameter values are passed to one routing program but never to the other. The following list describes in detail only the parameters that are significant to the dynamic routing program. Parameter values that are never passed to the dynamic routing program are not listed. For example, under the DYRFUNC parameter the value X'5' is not listed. X'5' is never passed to the dynamic routing program because it occurs only on a route initiate call to the distributed routing program.

If you use the same program as both a dynamic routing program and a distributed routing program, see "Parameters passed to the distributed routing program" on page 621 for descriptions of the parameters and values that are significant when using distributed routing calls.

DYRABCDE
Is the abend code returned when a routed transaction or program link request abends, or a Link3270 user transaction abends.

DYRABNLC
Is an abnormal event code, or null.

This parameter is significant when the dynamic routing program is invoked to stop a routed request. Any value other than null indicates that an abnormal event, other than a transaction abend, has occurred in the region to which the request was routed. Your routing program must not route further requests to the same region until the cause of the error has been investigated and fixed.

This field is used by CICSPlex System Manager. It is set by DB2 only. For more information, see Avoiding AEY9 abends in Developing applications.

DYRACMAA

This field applies to the routing of these items:

- Terminal-initiated transactions
- Transactions started by terminal-related START commands
- Program link (DPL) requests

For the routing of these types of requests, DYRACMAA contains one of these entries:

- The 31-bit address of the communications area (COMMAREA) of the application if the user application uses a COMMAREA or if you are using transaction routing, where the first transaction specifies either a COMMAREA or a channel on its **EXEC CICS RETURN** command
- The 31-bit address of the DFHROUTE container if the user application uses a channel and has created a container named DFHROUTE in the channel
- Null characters if the user application has no COMMAREA and no DFHROUTE container.

For the routing of all other types of requests, DYRACMAA contains null characters.

For the routing of the three types of eligible requests listed, if the user application uses a COMMAREA, the address depends on how the dynamic routing program is invoked.

- If your dynamic routing program is invoked for routing (DYRFUNC=0), the address of the input communications area, if one is available. In the same way, when your routing program is invoked because of a route-selection error (DYRFUNC=1) or for notification (DYRFUNC=3), the address is the address of the input communications area.
- If your routing program is invoked because a previously routed transaction or link request has ended normally (DYRFUNC=2), the address of the output communications area, if one is available. Routed applications can use their output communications area to pass information to the dynamic routing program.

If you are routing transactions and the user application uses a channel, the routing program is given the name instead of the address of the channel, which means that you cannot use the DYRCHANL parameter to inspect or change the contents of the containers.

When your routing program is invoked because the routed transaction abends (DYRFUNC=4), the information in the communications area, or in the DFHROUTE container, is not meaningful.

Your routing program can alter the data in the communications area of any application, or DFHROUTE container, addressed by DYRACMAA.

DYRACMAL

Applies to the routing of these items:

- Terminal-initiated transactions
- Transactions started by terminal-related START commands

- Program link (DPL) requests

For the routing of these types of requests, DYRACMAL contains one of the following numerical values:

- The length, in bytes, of the application COMMAREA if the user application uses a COMMAREA
- The length, in bytes, of the data in the DFHROUTE container if the user application uses a channel and has created a container named DFHROUTE in the channel
- Zero if the user application has no COMMAREA and no DFHROUTE container

For the routing of all other types of request, DYRACMAL contains zero.

DYRACTCMP
Is not used by the dynamic routing program. On invocation, it is set to nulls.

DYRACTID
Is not used by the dynamic routing program. On invocation, it is set to nulls.

DYRACTN
Is not used by the dynamic routing program. On invocation, it is set to nulls.

DYRAPPLICATION
Application context application name. It is set to nulls by default and if no application context is available.

DYRAPPLMAJOR
Application context application major version. It is set to nulls by default and if no application context is available.

DYRAPPLMICRO
Application context application micro version. It is set to nulls by default and if no application context is available.

DYRAPPLMINOR
Application context application minor version. It is set to nulls by default and if no application context is available.

DYRAPPLVER
Application context application version. It is set to nulls by default and if no application context is available.

DYRBLGTH
Is the length of the copy of the TIOA DFHLUC buffer.

This field applies only to dynamic transaction routing or to Link3270 requests (not to the routing of program link requests).

DYRBPNTR
Is the 31-bit address of a copy of the TIOA LUC buffer.

This field applies only to dynamic transaction routing and not to the routing of program link requests.

When your dynamic routing program is invoked for routing, because of a route-selection error (DYRFUNC=0), or for notification (DYRFUNC=3), it is given a copy of the input TIOA. Your routing program can alter the terminal input data passed to the routed transaction; see "Modifying the initial terminal data" on page 580.

When your routing program is invoked because a previously routed transaction has ended normally (DYRFUNC=2), it is given a copy of the output

TIOA. Your routing program can monitor the output TIOA to detect possible problems in the AOR; see "Receiving information from a routed transaction" on page 581.

When your routing program is called for a Link3270 bridge request (DYRTYPE=8), the address of a copy of the TIOA LUC buffer is not passed in DYRBPNTR.

DYRBRTK
Is the 8-byte bridge facility token associated with a Link3270 bridge request. This field is valid only when DYRTYPE=8.

DYRCABP
Indicates whether or not you want CICS to continue standard abend processing.

This field applies only to dynamic transaction routing, not to the routing of program link or Link3270 requests. If a linked-to program abends on a remote region, the abend is mirrored in the local region; that is, it is passed to the program that issued the EXEC CICS LINK command.

:

Y Continue with CICS abend processing.

N Stop the transaction, do not continue with CICS abend processing, and give control to the program specified by DYRLPROG.

You can use this option to pass control to a local program that can handle the condition in a way that you control and issue appropriate messages to terminal users.

If you enter N, you must ensure that DYRLPROG specifies the name of a valid program on the local system.

No default value applies to DYRCABP.

DYRCHANL
Is the name of the channel, if any, associated with the program link or START command. This field applies only to the routing of DPL requests, nonterminal-related START requests, and transactions started by terminal-related START requests. For other types of request, or if no channel is associated with the command, this field contains blanks.

Note that the routing program is given the *name* of the channel, not its address, and so is unable to use the contents of this field to inspect or change the contents of the containers. For information about how the routing program can inspect or change the contents of the application containers, see "Modifying the application's containers" on page 592 and the description of the DYRACMAA field.

DYRCLOUD
Application context cloud routing data. It is a container that encapsulates all of the other application context fields. It is set to nulls by default and if no application context is available.

DYRCOMP
Is the CICS component code. For calls to the dynamic routing program, it is always set to RT.

DYRCOUNT
Is a count of the times the dynamic routing program has been invoked for this

transaction or link request with DYRFUNC set to 0, 1, or 3. Use this field to limit the number of times your program tries to route a request.

DYRDTRRJ
Indicates whether the transaction, which is defined by the common transaction definition specified on the **DTRTRAN** system initialization parameter, is to be rejected or accepted for processing.

This field applies only to dynamic transaction routing and Link3270 request routing (not to the routing of program link requests), and is relevant only when DYRTRXN is set to Y.

The following values are valid:

Y The transaction is rejected. Y is the default.

N The transaction is not rejected.

This parameter is always set to the reject condition when the dynamic routing program is invoked. To dynamically route a transaction defined by the **DTRTRAN** system initialization parameter, you must change this indicator to the accept condition.

If you reject the transaction, message DFHAC2001, `Transaction` *`tranid`* `is unrecognized.` , is sent to the user's terminal for dynamic transaction routing. For Link3270 requests, the BRIH returned to the client contains a return code, indicating that the transaction was not found, and a compcode indicating that the routing program rejected the transaction specified on the **DTRTRAN** system initialization parameter.

DYRDTRXN
Indicates whether the transaction to be routed is defined by the common transaction definition specified on the DTRTRAN system initialization parameter or by a specific transaction definition.

This field applies only to dynamic transaction routing and Link3270 requests, not to the routing of program link requests.

The following values are valid:

Y The transaction is defined by the definition specified by the **DTRTRAN** system initialization parameter. That is, there is no resource definition for the input transaction identifier (ID).

For dynamic transaction routing, the transaction is started in the terminal-owning region using the transaction ID specified by the **DTRTRAN** system initialization parameter.

For dynamic transaction routing, the input transaction ID is passed to the dynamic routing program in the DYRTRAN field. For Link3270 requests, the common transaction definition is used to determine the routing characteristics of the request. The request still contains the original transaction ID, not the common transaction ID. If the request is run locally, the request is passed to the driver successfully, but the driver fails to start the user transaction because it is not defined.

N The transaction is not defined by the definition specified by the **DTRTRAN** system initialization parameter. An installed resource definition exists for the input transaction ID.

For dynamic transaction routing, the transaction is started in the terminal-owning region using the input transaction ID. The transaction ID passed to the dynamic routing program in the DYRTRAN field is

the remote transaction ID from the transaction resource definition (if this ID is different from the input transaction ID).

For Link3270 requests, the transaction ID passed to the routing program in the DYRTRAN field is the remote transaction ID defined in the TRANSACTION resource definition.

DYRERROR

Has a value only when DYRFUNC is set to 1. DYRERROR indicates the type of error that occurred during the last attempt to select a route. If an attempt to route over an IPIC connection failed and a subsequent attempt to use a connection of the same name also failed (for a reason other than SYSID not found), the type of error that occurred on the attempt to route over the connection is returned. The following values are valid:

0 The selected SYSID is unknown.

1 The selected system is not in service.

2 The selected system is in service, but no sessions are available.

3 An allocate request has been rejected, and SYSIDERR is returned to the application program. This error occurs for one of the following reasons:

- An XZIQUE global user exit program requested that the allocate be rejected
- CICS rejected the allocate request automatically because the QUEUELIMIT value specified on the CONNECTION resource definition was reached.

4 A queue of allocate requests has been purged, and SYSIDERR is returned to all the waiting application programs. This error occurs for one of the following reasons:

- An XZIQUE global user exit program requested that the queue be purged
- CICS purged the queue automatically because the MAXQTIME limit specified on the CONNECTION resource definition was reached.

5 The selected system does not support this function. This value occurs if the routing program tries to perform one of these actions:

- Route a transaction initiated by an **EXEC CICS START** command to a region that is not connected by an MRO or APPC parallel-session link.
- Route a transaction, or a program link or Link3270 request, across a LU6.1 connection.
- Route a Link3270 request to a region at an unsupported release of CICS.
- Route a transaction across an IPIC connection to a pre-CICS TS for z/OS, Version 4.1 region.
- Route an APPC device over an IPIC connection.

Values 6 - B all apply to attempts to route program link requests. For the meanings of these error conditions, see LINK in Reference -> Application development .

6 The **EXEC CICS LINK** command returned LENGERR.

7 The **EXEC CICS LINK** command returned PGMIDERR.

8 The **EXEC CICS LINK** command returned INVREQ.

9 The **EXEC CICS LINK** command returned NOTAUTH.

A The **EXEC CICS LINK** command returned TERMERR.

B The **EXEC CICS LINK** command returned ROLLEDBACK.

F The XPCERES global user exit program on the target region set a return code of UERCRESU, meaning that a required resource is unavailable on the target region. This error code is set for program link, Link3270 bridge, and non-terminal-related START requests.

DYRFUNC

Tells you the reason for this invocation of the dynamic routing program. The following values are valid:

0 Invoked for route selection.

1 Invoked because an error occurred in route selection.

2 Invoked because a previously routed transaction or program link request has ended successfully, or invoked for a request for which the user transaction ended successfully.

3 Invoked for notification of the destination of a statically routed request. This notification applies in the following cases:

ATI requests

A transaction defined as DYNAMIC(YES) has been initiated by a terminal-related automatic transaction initiation (ATI) request, for example, by the expiry of an interval control start request, but the transaction is ineligible for dynamic routing.

For information about which transactions initiated by terminal-related **EXEC CICS START** commands are eligible for dynamic routing, see Routing transactions invoked by START commands in Getting started.

Program link requests

The program is defined as DYNAMIC(YES), or is not defined, but the caller specified the name of a remote region on the SYSID option of the **EXEC CICS LINK** command.

In this case, specifying the target region explicitly takes precedence over any SYSID returned by the dynamic routing program.

Bridge requests

In session mode, the requested transaction is not the first user transaction and is defined as DYNAMIC(YES).

4 Invoked because the routed transaction or the requested user transaction abends.

7 Invoked to identify a call for end of unit of work processing

The DYRTYPE field tells you the type of routing or notification request.

DYRLEVEL

Is the level of CICS required in the target AOR to successfully process the routed request. The following values are valid:

X'00' Any currently supported version of CICS is able to process the request.

X'01' CICS TS for z/OS, Version 2.2. This value is set only for method requests for enterprise beans and CORBA stateless objects (handled by the distributed routing program).

X'02' CICS TS for z/OS, Version 2.3. This value is set only for method requests for enterprise beans and CORBA stateless objects (handled by the distributed routing program).

X'03' CICS TS for z/OS, Version 3.1. This value is set for these requests:
- DPL requests that have a channel associated with them.
- START requests that have a channel associated with them.
- Inbound Web services requests (handled by the distributed routing program).
- Method requests for enterprise beans and CORBA stateless objects (handled by the distributed routing program).

X'04' CICS TS for z/OS, Version 3.2.

Note that values greater than X'00' indicate the *specific*, not the minimum, level of CICS required to process the request successfully.

This parameter helps you to perform a "rolling upgrade" of a multi region logical server; one region at a time is upgraded from one release of CICS to the next, without bringing down the server. Requests that require a specific level of CICS can be routed to an appropriate AOR.This mixed level of operation, in which different CICS regions in the same logical server are at different levels of CICS, is for rolling upgrades only. It is not for permanent use, because it increases the risk of failure in some interoperability scenarios. The normal, recommended, mode of operation is that all the regions in a logical sever are at the same level of CICS and Java.

DYRLPROG

Is the name of the first program of the transaction to be routed or the name of the program specified on the link command to be routed.

Transaction routing

You can use this field to specify the name of an alternative program to be run if the transaction is routed locally. For example, if all remote CICS regions are unavailable, and the transaction cannot be routed, you might want to run a program in the local terminal-owning region to send an appropriate message to the user.

Do not set DYRLPROG to blanks when you specify DYRCABP=N. If you specify DYRCABP=N, ensure you also specify a valid program name on DYRLPROG.

Program link requests

When DYRFUNC is set to 0 or 3, DYRLPROG contains the name of the program to be linked, obtained using the following sequence:

1. From the REMOTENAME option of the installed program definition.
2. If REMOTENAME is not specified, or there is no program definition, from the PROGRAM option of the EXEC CICS LINK command.

You can use this field to specify that an alternative program, other than that named on the program link request, is to be linked. You can specify a local or remote program, depending on the region to which the request is to be routed.

Be aware that, if you change the value of DYRLPROG, and the alternative program you choose is defined as DYNAMIC(YES), the dynamic routing program is reinvoked for route selection.

Bridge requests

When DYRTYPE=8, do not change this field; any changes made are ignored by CICS.

You can change DYRLPROG on any call to the dynamic routing program, but it is effective only when DYRFUNC is set to 0 or 1.

DYRLUOW
The 8-byte local unit of work ID. This token forms part of the key for the LOCKED affinity type.

This field is valid only when DYRTYPE=4 or 9 (DPL) or DYRFUNC=7 (end of unit of work). DYRTYPE 4 is DPL without CHANNEL, DYRTYPE 9 is DPL with CHANNEL.

DYRNETNM
The netname of the CICS region identified in DYRSYSID.

If the DYRNETNM value is changed by the initial invocation of the dynamic routing program, CICS tries to route the request to the CICS region with the new netname.

DYRNUOW
The 27-byte network unit of work ID. This token forms part of the key for the LOCKED affinity type.

This field is valid only when DYRTYPE=4 or 9 (DPL) or DYRFUNC=7 (end of unit of work).

DYROPERATION
Application context operation name for the application entry point. It is a container that encapsulates all of the other application context fields. It is set to nulls by default and if no application context is available.

DYROPTER
Specifies whether the dynamic routing program is to be reinvoked when the routed transaction or link request ends (successfully or unsuccessfully). The following values are valid:

N The dynamic routing program is not to be reinvoked. N is the default.

Y The dynamic routing program is to be reinvoked.

You can specify this option for transactions, link requests, or bridge requests that are routed to remote CICS regions and also for those that are run locally.

DYRPLATFORM
Application context platform name for the platform where the application is deployed. It is a container that encapsulates all of the other application context fields. It is set to nulls by default and if no application context is available.

DYRPROCCMP
Is not used by the dynamic routing program. On invocation, it is set to nulls.

DYRPROCID
Is not used by the dynamic routing program. On invocation, it is set to nulls.

DYRPROCN
Is not used by the dynamic routing program. On invocation, it is set to nulls.

DYRPROCT
Is not used by the dynamic routing program. On invocation, it is set to nulls.

DYRPRTY
Can be used to set the dispatch priority of the task in the application-owning region, if the connection between the terminal-owning region and application-owning region is MRO or IPIC, or when processing a bridge request.

Transaction routing
Before invoking the dynamic routing program, CICS sets this value to the priority of the relay transaction task.

Program link requests
Before invoking the dynamic routing program, CICS sets this value to the priority of the task that issued the program link request.

Bridge requests
Before invoking the dynamic routing program, CICS set this value to the value defined in the TRANSACTION resource definition of the user transaction in the router region.

On return from the initial invocation of the dynamic routing program, if the DYRRTPRI value is Y and there is an MRO or IPIC connection between the terminal-owning region and application-owning region, CICS passes the DYPPRTY value to the application-owning region.

DYRQUEUE
Identifies whether or not the request is to be queued if no sessions are immediately available to the remote system identified by DYRSYSID. The following values are valid:

Y The request is to be queued if necessary. Y is the default.

N The request is not to be queued.

For bridge requests, DYRQUEUE is set to Y before the dynamic routing program is invoked. Any change made to this value by the user-replaceable program is ignored by CICS.

DYRRETC
Contains a return code that tells CICS how to proceed.

Transaction routing
The following values are valid:

0 Continue processing the transaction.

4 Stop the transaction without a message or abend.

8 Stop the transaction with either a message or an abend.

Whenever the routing program is invoked, DYRRETC is set to 0. When it is invoked for route selection or because an error occurs in route selection, if you want CICS to continue processing the transaction you must leave it set to 0.

To make CICS stop the transaction and issue a message or abend and return a value of 8.

To make CICS stop the transaction without issuing a message or abend (indicating that DFHDYP has done all the processing that is necessary), and return a value of 4.

Setting a return code of 4 for APPC transaction routing leads to unpredictable results, and should be avoided.

Setting any nonzero return code other than 4 is equivalent to setting 8.

Program link requests
The following values are valid:

0 Continue processing the link request.

Non-zero
Return an error condition to the program.

Whenever the routing program is invoked, DYRRETC is set to 0. When it is invoked for route selection or because an error occurs in route selection, if you want CICS to continue processing the link request, you must leave it set to 0.

To make CICS reject the link request, return a nonzero value. The program that issued the **EXEC CICS LINK** command receives a PGMIDERR condition, with a RESP2 value of 27.

Bridge requests
The following values are valid:

0 Continue processing the request.

4 Stop processing the request without issuing any error messages.

8 Stop processing the request with an error message.

Whenever the routing program is invoked, DYRRETC is set to 0. When it is invoked for route selection or because an error occurs in route selection, if you want CICS to continue processing the link request, you must leave it set to 0.

To make CICS stop the request without issuing a message, return a value of 4. The BRIH message header returned to the client contains a return code informing the client that the dynamic routing program has rejected the request, and a compcode that gives details of the reason why the last attempt to route the request failed.

To make CICS stop the request and issue a message, return a value of 8. The BRIH returned to the client contains a return code, informing the client that the dynamic routing program has rejected the request, and a compcode that gives details of the reason why the last attempt to route the request failed.

You do not set a return code when the routing program is invoked for notification or at transaction termination. Any code you set is ignored by CICS.

DYRRTPRI
Indicates whether or not the dispatch priority of the transaction, link request, or request is to be passed to the application-owning region, if the connection between the terminal-owning region and the application-owning region is MRO or IPIC. The following values are valid:

N The dispatch priority is not passed. N is the default.

Y The dispatch priority is passed.

DYRSRCTK
Is the MVS workload management service and reporting class token for the routed transaction. Your routing program must not alter this value, which is set by CICS and used by CICSPlex SM.

DYRSYSID
Is the system identifier (SYSID) of a CICS region. The exact meaning of this parameter depends on the values of DYRFUNC and DYRTYPE:

- When DYRFUNC is set to 0 (route selection):
  - If DYRTYPE is set to 0, 2, 3, or 8 (any type of transaction routing), DYRSYSID contains one of these names:

- The CICS region name specified on the REMOTESYSTEM option of the installed transaction definition
- If REMOTESYSTEM is not specified, the system name of the local CICS region

– If DYRTYPE is set to 4 or 9 (DPL routing), DYRSYSID contains one of these names:
- The CICS region name specified on the REMOTESYSTEM option of the installed program definition.

  Note: If the REMOTESYSTEM option names a remote region, the routing program cannot route the request locally.
- If REMOTESYSTEM is not specified, or there is no program definition, the system name of the local CICS region.

The dynamic routing program can accept the value of DYRSYSID or change it before returning to CICS.

If the SYSID you return to CICS is the same as the local SYSID, CICS runs the transaction or program in the local region.

- When DYRFUNC is set to 1 (route selection error), DYRSYSID contains the CICS region name returned to CICS by the dynamic routing program on its previous invocation.

  The action your dynamic routing program can take when DYRFUNC=1 depends on the DYRERROR parameter setting:
  – If DYRERROR is set to 0 (unknown SYSID) or 1 (CICS region not in service) and you want CICS to retry routing, you must change DYRSYSID before returning to CICS.
  – If DYRERROR is set to 2 (no session available) and you want CICS to retry routing, you must change DYRSYSID or change the value of DYRQUEUE to Y (queue the request until a session is available).
- When DYRFUNC is set to 2 (end of a routed request), DYRSYSID contains the name of the CICS region on which the completed transaction or link request ran.
- When DYRFUNC is set to 3 (notification):
  – For ATI requests, DYRSYSID contains one of these names:
    - The remote CICS region name specified on the SYSID option of the **EXEC CICS START** command
    - If SYSID is not specified, the remote CICS region name specified on the REMOTESYSTEM option of the installed transaction definition
    - If REMOTESYSTEM is not specified, the system name of the local CICS region.
  – For program link requests, DYRSYSID contains the remote CICS region name specified on the SYSID option of the EXEC CICS LINK command.
  – For bridge requests, DYRSYSID contains the SYSID of the CICS region where the request is routed and the user transaction run.

  Any changes to the values of DYRSYSID, or DYRNETNAME, are ignored.
- When DYRFUNC is set to 4 (abend), DYRSYSID contains the name of the CICS region on which the transaction abended.

DYRTRAN
Contains the remote transaction ID.

Transaction routing
When DYRFUNC is set to 0 or 3, DYRTRAN contains the remote transaction ID specified on the REMOTENAME option of the installed TRANSACTION resource.

Bridge requests
When DYRTYPE=8, DYRTRAN contains the transaction ID of the target user transaction because it is known in the current region. Note that this is not the same as the current transaction ID.

Program link requests
When DYRFUNC is set to 0 or 3, DYRTRAN contains the transaction ID of the remote mirror transaction, obtained using the following sequence:

1. From the TRANSID option on the LINK command.

   Note: A value specified on the TRANSID option of the LINK command cannot be overridden by the routing program.
2. From the TRANSID option on the program definition.
3. CSMI, the generic mirror transaction. CSMI is the default if neither of the TRANSID options are specified.

Your dynamic routing program can accept this remote transaction ID, or supply a different transaction name for forwarding to the remote CICS region. If the supplied name is longer than four characters, it is truncated by CICS.

You can change DYRTRAN on any call to the dynamic routing program, but the change is effective only in these circumstances:

- When DYRFUNC is set to 0 or 1.
- If the original value was not obtained from the TRANSID option of an **EXEC CICS LINK** command. A value specified on the TRANSID option of a LINK command cannot be overridden by the routing program.

DYRTYPE
Is the type of routing request for which the program is being invoked. For transaction routing, this field is meaningful only when DYRFUNC is set to 0 (route selection) or 3 (notify). These values can be passed to the dynamic routing program:

0 A transaction started from a terminal.

1 An ATI request that is to be statically routed.

2 A transaction started by a terminal-related **EXEC CICS START** command, where there is no data and no channel associated with the START.

3 A transaction started by a terminal-related **EXEC CICS START** command, where there is data but no channel associated with the START.

4 A program link request without a channel.

8 A bridge request.

9 A program link request with a channel.

A A transaction started by a terminal-related **EXEC CICS START** command, where there is a channel associated with the START.

DYRUAPTR
If DYRVER is 7 or greater, this field contains the address of the new user area, DYRUSERN. The new user area mechanism makes the source of the routing

program independent of the CICS release that created the communications area. The old user area field DYRUSER is retained only for compatibility purposes.

The user area can be mapped with the DYRUAREA DSECT.

In systems where DYRUAPTR is less than 7, the contents of DYRUAPTR are unpredictable.

DYRUOWAF
This field is used by the called user exit application to inform CICS that a FUNC=7 callback is required for DYRTPE=4 or 9 (DPL) requests when the current network unit of work completes.

The field contains no relevant data after the call from CICS is received DYRUOWAF is used only to provide a response to CICS for DPL requests. The following values are valid:

N Callback is not required.

Y Callback is required.

In the case of multiple DPL calls for a UOW, if any of the calls return Y, callback occurs at end of the UOW.

DYRUSER
Is a 1024-byte user area.

This field is retained only for compatibility purposes; see the descriptions of the DYRUAPTR and DYRUSERN fields.

DYRUSERID
Is the CICS user ID associated with the request.

For transaction routing, program link requests, and bridge requests, DYRUSERID contains the user ID under which the current transaction is running.

By examining this field when it is invoked for routing or because of a route-selection error (DYRFUNC=0 or 1, respectively), your routing program can route requests based on the user ID associated with the request.

DYRUSERN
Is a 1024-byte user area.

CICS initializes this user area to zeros before invoking the dynamic routing program for a given task. This user area can be modified by the dynamic routing program; the modified area is passed to subsequent invocations of the dynamic routing program for the same request.

DYRVER
Is the version number of the dynamic routing program interface. For CICS Transaction Server for z/OS, Version 5 Release 2, the number is 11.

Naming your dynamic routing program

The supplied, user-replaceable dynamic routing program is named DFHDYP. If you write your own version, you can name it differently.

Procedure

1. Identify the name of the dynamic routing program by using the EXEC CICS INQUIRE SYSTEM command. The DTRPROGRAM field contains the name of the current program.

2. Change the name of the dynamic routing program using either of the following methods:
   - Change the value of the DTRPGM system initialization parameter.
   - Use the SET SYSTEM DTRPROGRAM command.
3. Create a new PROGRAM resource for your customized dynamic routing program.

Results

CICS uses your customized dynamic routing program instead of the supplied program, DFHDYP.

Testing your dynamic routing program

You can use the CICS execution diagnostic facility (EDF) to test your dynamic routing program. To do so, you must name your program something other than DFHDYP, because you cannot use EDF for programs that begin with "DFH". For details of how to use EDF, see Execution diagnostic facility (EDF), in the *CICS Application Programming Guide*.

You can use EDF in either single- or dual-terminal mode. If you choose single-terminal mode, EDF displays screens for both the dynamic routing program and the application program that is invoked by the routed transaction. The screens relate to:

- The initial invocation of the dynamic routing program for route selection or notification (DYRFUNC=0 or DYRFUNC=3)
- The invocation of the dynamic routing programm if an error occurs in route selection (DYRFUNC=1)
- The invocation of the application program
- The termination of the task
- The invocation of the dynamic routing program at termination of the routed transaction or link request (DYRFUNC=2), if you have specified DYROPTER=Y
- The invocation of the dynamic routing program if the routed transaction abends (DYRFUNC=4), if you have specified DYROPTER=Y.

If you want EDF to display the execution of your dynamic routing program only, either choose dual-terminal mode, or use one of the other methods described in Execution diagnostic facility (EDF), in the *CICS Application Programming Guide*.

Dynamic transaction routing sample programs

You can use the CICS-supplied sample dynamic routing program, DFHDYP, or you can write your own in COBOL, PL/I, C, or assembler language. You can also change the name of the program.

The CICS-supplied sample dynamic routing program is named DFHDYP. The corresponding copy book that defines the communications area is DFHDYPDS. There are assembler-language, COBOL, PL/I, and C source-level samples and copy books. The supplied programs and copy books, and the libraries in which they can be found, are summarized in Table 32 on page 608.

Table 32. Dynamic transaction routing programs and copy books

Language	Member name	Library
Programs: Assembler COBOL PL/I C	DFHDYP DFHDYP DFHDYP DFHDYP	SDFHSAMP SDFHCOB SDFHPL1 SDFHC370
Copy books: Assembler COBOL PL/I C	DFHDYPDS DFHDYPDS DFHDYPDS DFHDYPDS	SDFHMAC SDFHCOB SDFHPL1 SDFHC370

When invoked with DYRFUNC set to '0', the sample programs accept the sysid and remote transaction name that are passed in fields DYRSYSID and DYRTRAN of the communications area, and set DYRRETC to '0' before returning to CICS. When invoked with DYRFUNC set to '2' or '3', they set a return code of '0'. When invoked with DYRFUNC set to '1' or '4', they set a return code of '8'.

If you want to route transactions or DPL requests dynamically, you must customize DFHDYP, replace it completely with your own routing program, or use CICSPlex System Manager.

Chapter 20. Writing a distributed routing program

You can use a distributed routing program to route different types of request in CICS, including inbound web services and non-terminal START requests.

The distributed routing program is named on the **DSRTPGM** system initialization parameter in the routing and target CICS regions. You can use a distributed routing program to route these requests:

- CICS business transaction services (BTS) processes and activities
- Non-terminal-related **EXEC CICS START** requests.

  For information about which non-terminal-related START requests are eligible for distributed routing, see Routing transactions invoked by START commands in Getting started.
- Inbound web service requests

You cannot use the distributed routing program to route these requests:

- Transactions initiated from user terminals
- Transactions initiated by terminal-related **EXEC CICS START** commands
- Program-link requests

To route these requests, you must use the dynamic routing program named on the **DTRPGM** system initialization parameter. How to write a dynamic routing program is described in Chapter 19, “Writing a dynamic routing program,” on page 575. The dynamic routing program and the distributed routing program can be the same program.

If you use CICSPlex System Manager (CICSPlex SM) to manage your CICSplex, you can use the routing program, EYU9XLOP. This program supports workload balancing and workload separation. You can define which regions in the CICSplex can participate in the workload, and define any transaction affinities that govern the regions to which particular requests must be routed. For more information about workload management in CICSPlex SM, see Configuring workload management in Configuring.

Differences between the distributed and dynamic routing interfaces

The distributed routing interface differs from the dynamic routing interface in several significant respects.

If you have previously written a dynamic routing program, and are about to write a distributed routing program, bear in mind that:

1. The dynamic routing program and the distributed routing program are invoked if the resource (the transaction or program) is defined as DYNAMIC(YES). The exception is for BTS activities that are run asynchronously, for which the distributed routing program is invoked even if the associated transaction is defined as DYNAMIC(NO). In this situation, the distributed routing program cannot route the request, but it can monitor the effect of the request on workloads, or perform other activities. What this means is that the distributed routing program is better able to monitor the effect of statically-routed requests on the relative workloads of the target regions.

2. Because the dynamic routing program uses the hierarchical "hub" routing model—one routing program controls access to resources on several target regions—*the routing program that is invoked at termination of a routed request is the same program that was invoked for route selection*.

   The distributed routing program, on the other hand, uses the distributed model, which is a peer-to-peer system; the routing program itself is distributed. *The routing program that is invoked at initiation, termination, or abend of a routed transaction is not the same program that was invoked for route selection*—it is the routing program on the target region.

   Because the dynamic routing program is invoked only on the routing region, the order of its invocations is strictly defined:

   a. Route selection or notification
   b. Route selection error (if appropriate, and possibly repeated)
   c. Transaction termination or abend (if requested).

3.

   For a single request, the user area passed to each invocation of the dynamic routing program is the same piece of storage; any modifications made to the user area on one invocation are retained and passed to the next invocation.

   The distributed routing program, on the other hand, may be invoked on the target region as well as on the routing region; because of this, the order of its invocations is less strictly defined. For example, the final invocation on the routing region (for "routing attempt complete") may occur before or after the first invocation on the target region (for "transaction initiation"). To cope with this uncertainty, the user area passed to the distributed routing program on its first invocation on the target region is a *copy* of the user area on the routing region. This means that any modifications to the user area made on the target region have no effect on the user area in the routing region. For more details, see the description of the DYRUSER field in "Parameters passed to the distributed routing program" on page 621.

4. The distributed routing program is invoked at more points than the dynamic routing program. "When the distributed routing program is invoked" on page 615 explains the points at which the distributed routing program is invoked, and the region on which each invocation occurs.

5. Unlike the dynamic routing program, the distributed routing program **cannot**:

   - Select a target region by supplying a netname (any value set in the DYRNETNM field of the communications area is ignored). The target must be specified by its CICS system identifier (sysid).
   - Change the remote transaction name passed to the target region. (Any value set in the DYRTRAN field of the communications area is ignored.)
   - Change the initial program associated with a routed request. (Any value set in the DYRLPROG field of the communications area is ignored).
   - Choose that the request is not to be queued if there are no MRO sessions to the target region. (The DYRQUEUE field of the communications area is always set to 'Y'.)
   - Modify the routed application's communications area. (The routing program is not passed the address of the routed application's communications area in field DYRACMAA.)
   - Pass the dispatch priority of the transaction to the target region. (The DYRRTPRI field of the communications area is always set to 'N'.)

   These restrictions are documented more fully in the descriptions of the relevant fields in the DFHDYPDS communications area.

Routing BTS activities

You can use a distributed routing program to dynamically route CICS business transaction services (BTS) processes and activities.

Which BTS activities can be dynamically routed?

Not all activations of BTS processes and activities can be routed.

Processes and activities that are activated asynchronously with the requestor—by means of a RUN ASYNCHRONOUS command—can be routed either dynamically or statically.

Processes and activities that are activated synchronously with the requestor—by means of a RUN SYNCHRONOUS or LINK command—are always run locally. They cannot be routed, *neither dynamically nor statically*. A RUN SYNCHRONOUS or LINK command issued against an activity whose associated transaction is defined as DYNAMIC(YES), or as residing on a remote region, results in the activity being run locally.

Thus, to be eligible for dynamic routing:

1. A BTS process or activity must be run asynchronously with the requestor, by means of a RUN ASYNCHRONOUS command.
2. The TRANSACTION definition for the transaction associated with the process or activity must specify DYNAMIC(YES).

"Daisy-chaining" is not supported. That is, once a BTS activity has been routed to a target region it cannot be re-routed from the target to a third region, even though its associated transaction is defined as DYNAMIC(YES).

When the distributed routing program is invoked

For BTS processes and activities started by RUN ASYNCHRONOUS commands, CICS invokes the distributed routing program at the following points:

On the routing region:

1. Either of the following:
   - For routing the activity. This occurs when the transaction associated with the activity is defined as DYNAMIC(YES).
   - For notification of a statically-routed request. This occurs when the transaction associated with the activity is defined as DYNAMIC(NO). The routing program is not able to route the activity. It could, however, do other things.
2. If an error occurs in route selection—for example, if the target region returned by the routing program on the route selection call is unavailable. This gives the routing program the opportunity to specify an alternate target. This process iterates until the routing program selects a target that is available or sets a non-zero return code.
3. After CICS has tried (successfully or unsuccessfully) to route the activity to the target region.

   This invocation signals that (unless the routing region and the target region are one and the same) the routing region's responsibility for this transaction has been discharged. The routing program might, for example, use this invocation to release any resources that it has acquired on behalf of the transaction.

On the target region:

These invocations occur only if the target region is CICS TS for OS/390®, Version 1.3 or later and the routing program on the routing region has specified that it should be reinvoked on the target region:

1. When the activation starts on the target region (that is, when the transaction that implements the activity starts).
2. If the routed activation (transaction) ends successfully.
3. If the routed activation (transaction) abends.

Figure 69 shows the points at which the distributed routing program is invoked, and the region on which each invocation occurs. Note that the "target region" is not necessarily remote—it could be the local (routing) region, if the routing program chooses to run the activity locally.

Figure 69. When and where the distributed routing program is invoked

Related information:

About workload view route fields

Changing the target CICS region

The DYRSYSID field of the communications area passed to the distributed routing program initially contains the system identifier (sysid) of the default target region to which the process or activity is to be routed. This is derived from the value of the REMOTESYSTEM option of the installed transaction definition on the routing region. If REMOTESYSTEM is not specified, the sysid passed is that of the local CICS region.

When it is invoked for route selection, the distributed routing program can change the target region by changing the value in DYRSYSID.

If the specified sysid is invalid, or cannot be found, SYSIDERR is returned to the distributed routing program—which may deal with the error by returning a different sysid—see "If an error occurs in route selection" on page 613.

If the routing program changes the sysid when it is invoked for notification, routing complete, transaction initiation, transaction termination, or abend, the change has no effect.

Telling CICS whether to route the activity

When the routing program is invoked for routing, if you want the process or activity to be routed (whether you have changed any values or not) return a zero value to CICS in field DYRRETC of the communications area.

When you return control to CICS with return code zero, CICS first compares the returned sysid with its own local sysid:

- If the sysids are the same (or the returned sysid is blank) CICS executes the RUN request locally. When it executes the request locally, CICS writes message DFHSH0102 to the CSSH transient data queue.
- If the two sysids are not the same, CICS routes the request to the remote CICS region.

If you want CICS to treat the request as *unserviceable*, return a non-zero value. For information about unserviceable requests, see Dealing with unserviceable requests in Troubleshooting and support.

Returning a value in DYRRETC has no effect when the routing program is invoked for notification, routing complete, transaction initiation, transaction termination, or abend.

If an error occurs in route selection

If an error occurs in route selection—for example, if the sysid returned by the distributed routing program is unavailable or unknown—the distributed routing program is invoked again.

When the program is re-invoked, you have a choice of actions:

1. You can try to route the request to a different target region, by changing the sysid, and issuing a return code of '0' in DYRRETC.

   If this region too is unavailable, the routing program is again invoked for a route selection error. A count of the times the routing program has been invoked for routing purposes for this request is passed in field DYRCOUNT. Use this count to help you decide when to stop trying to route the request.
2. You can tell CICS to treat the request as "unserviceable", by issuing a non-zero return code in DYRRETC.

   Sometimes, perhaps because of a transaction affinity, it is essential that an activity should execute on a particular target region, and on no other. If this is the case, and the target region is unavailable, classify the request as unserviceable. Instead of reinvoking the routing program for a route selection error, CICS:

   a. Tries repeatedly to route the request to the specified target region, at 1-minute intervals.

      If one of these attempts is successful, CICS issues message DFHSH0108. The routing program is invoked on the routing region for "routing attempt complete", and, if specified, on the target region for "transaction initiation".

   b. Every hour, if the target region is still unavailable, issues message DFHSH0106.

   c. If the target region is still unavailable 24 hours after the request was issued, issues message DFHSH0107, and stops trying to route the request, which is discarded. The routing program is invoked on the routing region for "routing attempt complete".

Invoking the distributed routing program on the target region

The route selection, notification, route selection error, and routing complete invocations of the distributed routing program all occur on the routing region. If you want the routing program to be re-invoked on the target region, set the DYROPTER field in the communications area to 'Y'. You must do this on the program's initial (route selection or notification) invocation—and again, if it is reinvoked for a route selection error.

If the routing program sets DYROPTER to 'Y', it is re-invoked on the target region:

- When the activation is about to be initiated on the target region
- If the routed activation (transaction) terminates successfully
- If the routed activation (transaction) abends.

Each time it is invoked on the target region, the routing program could update a count of BTS activities that are currently running on that region. When it is invoked for routing, the routing program could use the counts maintained by all the regions in the routing set (including itself) as input to its routing decision. This requires that each region in the routing set has access to a common data set on which the counts are recorded.

Routing non-terminal-related START requests

You can use a distributed routing program to dynamically route non-terminal-related **EXEC CICS START** requests.

Which requests can be dynamically routed?

For a non-terminal-related START request to be eligible for dynamic routing, all of the following conditions must be met:

- The request is eligible for *enhanced* routing. For general information about the "enhanced" method of routing transactions invoked by EXEC CICS START commands, and for specific information about which non-terminal-related START requests are eligible for enhanced routing, see Routing transactions invoked by START commands, in the *CICS Intercommunication Guide*.
- The transaction definition in the routing region specifies both ROUTABLE(YES) and DYNAMIC(YES).
- The SYSID option of the START command does not specify the name of a remote region. (That is, the remote region on which the transaction is to be started must not be specified explicitly.)

If the request is fully eligible for dynamic routing, the distributed routing program is invoked for routing. The START request is function-shipped to the target region returned by the routing program.

Note:

1. If the request is ineligible for enhanced routing, the distributed routing program is not invoked. Unless the SYSID option of the START command specifies a remote region explicitly, the START request is function-shipped to the target region named in the REMOTESYSTEM option; if REMOTESYSTEM is not specified, the START executes locally.
2. If the request is eligible for enhanced routing but not for dynamic routing (the transaction may, for example, be defined as DYNAMIC(NO)) the distributed routing program is invoked for notification only—it cannot route the request. Unless the SYSID option of the START command specifies a remote region

explicitly, the START request is function-shipped to the target region named in the REMOTESYSTEM option; if REMOTESYSTEM is not specified, the START executes locally.

"Daisy-chaining" is not supported. That is, once a non-terminal-related START request has been dynamically routed to a target region it cannot be dynamically routed from the target to a third region, even though the transaction is defined as ROUTABLE(YES) and DYNAMIC(YES). The transaction may, however, be *statically* routed from the target region to a third region.

For definitive information about which non-terminal-related START requests are eligible for dynamic routing, see Non-terminal-related START commands, in the *CICS Intercommunication Guide*.

When the distributed routing program is invoked

For non-terminal-related START requests that are eligible for enhanced routing, CICS invokes the distributed routing program at the following points:

On the routing region:

1. Either of the following:
   - For routing the request.
   - For notification of a statically-routed request. This occurs when a transaction defined as ROUTABLE(YES) is eligible for *enhanced* routing but not for *dynamic* routing because one or both of the following applies:
     - The transaction definition specifies DYNAMIC(NO).
     - The SYSID option of the START command names a remote region explicitly.

     The routing program is not able to route the request. It could, however, do other things.
2. If an error occurs in route selection—for example, if the target region returned by the routing program on the route selection call is unavailable. This gives the routing program the opportunity to specify an alternate target. This process iterates until the routing program selects a target that is available or sets a non-zero return code.
3. After CICS has tried (successfully or unsuccessfully) to route the request to the target region.

   This invocation signals that (unless the routing region and the target region are one and the same) the routing region's responsibility for this transaction has been discharged. The routing program might, for example, use this invocation to release any resources that it has acquired on behalf of the transaction.

On the target region:

These invocations occur only if the target region is CICS TS for OS/390, Version 1.3 or later and the routing program on the routing region has specified that it should be reinvoked on the target region:

1. When the transaction associated with the request starts on the target region.
2. If the transaction ends successfully.
3. If the transaction abends.

Figure 70 on page 616 shows the points at which the distributed routing program is invoked, and the region on which each invocation occurs. Note that the "target region" is not necessarily remote—it could be the local (routing) region, if the

routing program chooses to execute the START request locally.

Figure 70. When and where the distributed routing program is invoked

Changing the target CICS region

The DYRSYSID field of the communications area passed to the distributed routing program initially contains the system identifier (sysid) of the default target region to which the request is to be routed. This is derived from the value of the REMOTESYSTEM option of the installed transaction definition on the routing region. If REMOTESYSTEM is not specified, the sysid passed is that of the local CICS region.

When it is invoked for route selection, the distributed routing program can change the target region by changing the value in DYRSYSID.

If the specified sysid is invalid, or cannot be found, SYSIDERR is returned to the distributed routing program—which may deal with the error by returning a different sysid—see "If an error occurs in route selection" on page 617.

If the routing program changes the sysid when it is invoked for notification, routing complete, transaction initiation, transaction termination, or abend, the change has no effect.

Telling CICS whether to route the request

When the routing program is invoked for routing, if you want the request to be routed (whether you have changed any values or not) return a zero value to CICS in field DYRRETC of the communications area.

When you return control to CICS with return code zero, CICS first compares the returned sysid with its own local sysid:

- If the sysids are the same CICS executes the request locally.
- If the two sysids are not the same, CICS routes the request to the remote CICS region.

If you want CICS to reject the START request, return a non-zero value. The EXEC CICS START command receives a SYSIDERR condition, with a RESP2 value indicating that the START request has been rejected by the routing program.

Returning a value in DYRRETC has no effect when the routing program is invoked for notification, routing complete, transaction initiation, transaction termination, or abend.

If an error occurs in route selection

If an error occurs in route selection—for example, if the sysid returned by the distributed routing program is unavailable or unknown—the routing program is invoked again.

When an the routing program is re-invoked, you have a choice of actions:

1. You can try to route the request to a different target region, by changing the sysid, and issuing a return code of ‘0' in DYRRETC.

   If this region too is unavailable, the routing program is again invoked for a route selection error. A count of the times the routing program has been invoked for routing purposes for this request is passed in field DYRCOUNT. Use this count to help you decide when to stop trying to route the request.
2. You can tell CICS not to continue trying to route the request, by issuing a non-zero return code in DYRRETC.

Using the XICERES exit to check the availability of resources on the target region

You can use an XICERES global user exit program to check that all resources required by the started transaction are available on the target region.

The XICERES exit is invoked, if enabled, on the target region before CICS processes a dynamically-routed START request.

If, for example, the transaction to be started is disabled on the target region, or a required file is missing, your exit program can give the distributed routing program the opportunity to route the request to a different region. To do this, it should set a return code of UERCRESU. This causes CICS to:

1. Return a RESUNAVAIL condition on the EXEC CICS START command executed by the mirror on the target region. (This condition is *not* returned to the application program.)
2. Set the DYRERROR field of the routing program's communications area to 'F'—resource unavailable.
3. Reinvoke the routing program, on the routing region, for route selection failure—see “If an error occurs in route selection.”

For information about writing an XICERES global user exit program, see The XICERES global user exit.

If a required resource is unavailable on the target region, but the XICERES exit is unavailable or disabled (or is enabled but does not set the UERCRESU return code), the client program receives an error response.

Invoking the distributed routing program on the target region

The route selection, notification, route selection error, and routing complete invocations of the distributed routing program all occur on the routing region. If you want the routing program to be re-invoked on the target region, set the DYROPTER field in the communications area to 'Y'. You must do this on the program's initial (route selection or notification) invocation—and again, if it is reinvoked for a route selection error.

If the routing program sets DYROPTER to 'Y', it is re-invoked on the target region:

- When the transaction associated with the routed request is about to be initiated on the target region
- If the transaction terminates successfully
- If the transaction abends.

Each time it is invoked on the target region, the routing program could update a count of transactions that are currently running on that region. When it is invoked for routing, the routing program could use the counts maintained by all the regions in the routing set (including itself) as input to its routing decision. This requires that each region in the routing set has access to a common data set on which the counts are recorded.

Routing inbound web service requests

You can use a distributed routing program to dynamically route inbound web service requests.

When the distributed routing program is invoked

For inbound Web service requests that are eligible for enhanced routing, CICS invokes the distributed routing program at the following points:

On the routing region:

1. Either of the following:
   - For routing the request.
   - For notification of a statically-routed request. This occurs when the transaction definition specifies DYNAMIC(NO).

     The routing program is not able to route the request. It could, however, do other things.
2. If an error occurs in route selection—for example, if the target region returned by the routing program on the route selection call is unavailable. This gives the routing program the opportunity to specify an alternate target. This process iterates until the routing program selects a target that is available or sets a non-zero return code.
3. After CICS has tried (successfully or unsuccessfully) to route the request to the target region.

   This invocation signals that (unless the routing region and the target region are one and the same) the routing region's responsibility for this transaction has been discharged. The routing program might, for example, use this invocation to release any resources that it has acquired on behalf of the transaction.

On the target region:

1. When the transaction associated with the request starts on the target region.
2. If the transaction ends successfully.
3. If the transaction abends.

Figure 71 on page 619 shows the points at which the distributed routing program is invoked, and the region on which each invocation occurs. Note that the “target region” is not necessarily remote—it could be the local (routing) region, if the routing program chooses to execute the request locally.

Figure 71. When and where the distributed routing program is invoked

Changing the target CICS region

The DYRSYSID field of the communications area passed to the distributed routing program initially contains the system identifier (sysid) of the default target region to which the request is to be routed. This is derived from the value of the REMOTESYSTEM option of the installed transaction definition on the routing region. If REMOTESYSTEM is not specified, the sysid passed is that of the local CICS region.

When it is invoked for route selection, the distributed routing program can change the target region by changing the value in DYRSYSID.

If the specified sysid is invalid, or cannot be found, SYSIDERR is returned to the distributed routing program—which may deal with the error by returning a different sysid—see "If an error occurs in route selection" on page 620.

If the routing program changes the sysid when it is invoked for notification, routing complete, transaction initiation, transaction termination, or abend, the change has no effect.

Telling CICS whether to route the request

When the routing program is invoked for routing, if you want the request to be routed (whether you have changed any values or not) return a zero value to CICS in field DYRRETC of the communications area.

When you return control to CICS with return code zero, CICS first compares the returned sysid with its own local sysid:

- If the sysids are the same CICS executes the request locally.
- If the two sysids are not the same, CICS routes the request to the remote CICS region.

If you want CICS to reject the request, return a non-zero value.

Returning a value in DYRRETC has no effect when the routing program is invoked for notification, routing complete, transaction initiation, transaction termination, or abend.

If an error occurs in route selection

If an error occurs in route selection—for example, if the sysid returned by the distributed routing program is unavailable or unknown—the routing program is invoked again.

When the routing program is re-invoked, you have a choice of actions:

1. You can try to route the request to a different target region, by changing the sysid, and issuing a return code of ‘0' in DYRRETC.

   If this region too is unavailable, the routing program is again invoked for a route selection error. A count of the times the routing program has been invoked for routing purposes for this request is passed in field DYRCOUNT. Use this count to help you decide when to stop trying to route the request.
2. You can tell CICS not to continue trying to route the request, by issuing a non-zero return code in DYRRETC.

Invoking the distributed routing program on the target region

The route selection, notification, route selection error, and routing complete invocations of the distributed routing program all occur on the routing region. However, you can re-invoke the routing program on the target region.

To do so, set the DYROPTER field in the communications area to 'Y'. You must do this on the program's initial (route selection or notification) invocation—and again, if it is reinvoked for a route selection error.

If the routing program sets DYROPTER to 'Y', it is re-invoked on the target region:

- When the transaction associated with the routed request is about to be initiated on the target region
- If the transaction terminates successfully
- If the transaction abends.

Each time it is invoked on the target region, the routing program could update a count of transactions that are currently running on that region. When it is invoked for routing, the routing program could use the counts maintained by all the regions in the routing set (including itself) as input to its routing decision. This requires that each region in the routing set has access to a common data set on which the counts are recorded.

Routing by user ID

Optionally, your routing program can route requests based on the CICS user ID (userid) associated with the request. The DYRUSERID field of the communications area contains the user ID. When it is invoked for routing or because of a route-selection error, your routing program can base its routing decision on the contents of this field.

For details of how the userid is set for different types of request, see the description of the DYRUSERID field in “Parameters passed to the distributed routing program” on page 621.

Dealing with an abend on the target region

If a routed request fails on the target region, CICS invokes the routing program for transaction abend, returning the abend code in field DYRABCDE of the communications area.

This invocation occurs on the target region, and only if the routing program has specified, on a previous call on the routing region, that it should be reinvoked on the target region.

The recommended way of dealing with an abend on the target region is as follows:

1. Code your routing program so that, on each route selection (and route selection error) call, it specifies that it is to be reinvoked (for transaction initiation, termination, and abend) on the target region.
2. If the routing program is invoked, on the target region, for transaction abend, it conveys full details of the failed request to the routing region. It could, for example, write the communications area to a shared resource, such as an RLS file or a shared data table.
3. The routing program on the routing region checks the shared resource at predetermined intervals.
4. When the routing program on the routing region discovers that a routed request has failed, it performs the following steps:
   a. Removes the target region from its routing set.
   b. Retries the request on another region. It tries repeatedly until either the request is successful or all possible AORs have been tried unsuccessfully. In the latter case, it returns an error response to the client.

Link checks and information for distributed routing programs

When you write a distributed routing program, you can check that a link is available before routing your request. You can also keep information about how requests are routed.

- When writing your routing program, you can use the CICS Explorer **ISC/MRO Connections** operations view or the **EXEC CICS INQUIRE CONNECTION** and **INQUIRE IRC** commands to confirm that a link is available before routing a request. The **EXEC CICS INQUIRE** and **EXEC CICS SET** commands are described in System commands in Reference > System programming.
- Because the distributed routing program runs outside a unit of work environment, your program must not alter any recoverable resources, or issue file control or temporary storage requests.
- If you want to keep information about how requests are routed, it must be done in the user routing program, perhaps by writing the information to a data set. Because the routing program is distributed, all the CICS regions in the transaction routing set must have access to the data set.
- The distributed routing program can be RMODE ANY but must be AMODE 31.

Parameters passed to the distributed routing program

A number of parameters are passed to the distributed routing program. The communications area is mapped by the copy book DFHDYPDS, which is in the appropriate CICS library for all the supported programming languages.

The same communications area is passed to both the distributed routing program and the dynamic routing program. Some parameters are meaningful to one routing program but not to the other. Some parameter values are passed to one routing program but never to the other. The following list describes in detail only the parameters that are significant to the distributed routing program; parameter values that are never passed to the distributed routing program are not listed. For example, under the **DYRTYPE** parameter the value X'4' is not listed because it is never passed to the distributed routing program; it occurs only on a program link-related call to the dynamic routing program.

If you use the same program as both a distributed routing program and a dynamic routing program, for descriptions of the parameters and values that are significant on dynamic routing calls refer to "Parameters passed to the dynamic routing program" on page 593.

DYRABCDE
Is the abend code returned when the transaction associated with a routed request abends in the target region.

This field is significant when the distributed routing program is invoked to stop a routed request. Any value other than blanks indicates that the transaction has abended in the target region (which, for the distributed routing program, is also the region in which the routing program is invoked).

For general information about how to handle other types of abend, see "Dealing with an abend on the target region" on page 621.

DYRABNLC
Is an abnormal event code, or null.

This field is significant when the distributed routing program is invoked to stop a routed request. Any value other than null indicates that an abnormal event, *other than a transaction abend*, has occurred in the region to which the request was routed (which, for the distributed routing program, is also the region in which the routing program is invoked). Your routing program must not route further requests to the same region until the cause of the error has been investigated and fixed.

This field is for use by CICSPlex System Manager. Currently, it is set only by DB2. For more information, see Avoiding AEY9 abends in Developing applications.

DYRACMAA
Is not used by the distributed routing program. On invocation, it is set to zeros.

DYRACMAL
Is not used by the distributed routing program. On invocation, it is set to zeros.

DYRACTCMP
Indicates whether the BTS activity is completing. When a process is being routed, DYRACTCMP indicates whether the root activity is completing.

This field applies only to the routing of BTS processes and activities. Its contents are significant on calls to stop a transaction.

These values are possible:

Y The root activity is the final activation of the BTS activity.

N The root activity is not the final activation of the BTS activity.

DYRACTID
Is the CICS-assigned, 52-character activity identifier of the BTS activity being routed. When a process is being routed, DYRACTID returns the identifier of the root activity.

This field applies only to the routing of BTS processes and activities.

DYRACTN
Is the name of the BTS activity being routed. When a process is being routed, DYRACTN returns the name of the root activity; that is, DFHROOT.

This field applies only to the routing of BTS processes and activities.

DYRBLGTH
Is not used by the distributed routing program. On invocation, it is set to zeros.

DYRBPNTR
Is not used by the distributed routing program. On invocation, it is set to zeros.

DYRCABP
Indicates whether you want CICS to continue standard abend processing.

This field is not used by the distributed routing program. On invocation, it is set to Y.

DYRCHANL
Is the name of the channel, if any, associated with the program link or START command. This field applies only to the routing of DPL requests, nonterminal-related START requests, and transactions started by terminal-related START requests. For other types of request, or if no channel is associated with the command, this field contains blanks.

Note that the routing program is given the *name* of the channel, not its address, and so is unable to inspect or change the contents of the containers.

DYRCOMP
Is the CICS component code. For calls to the distributed routing program, it is set to one of the following:

SH Scheduler services domain. For routing of BTS processes and activities and for nonterminal-related START requests.

RZ Request streams domain. For routing of method requests for inbound web service requests.

DYRCOUNT
Is a count of the times the distributed routing program has been invoked for this request with DYRFUNC set to 0, 1, or 3. Use this field to limit the number of times your program tries to route a request.

DYRDTRRJ
Indicates whether the transaction, which is defined by the common transaction definition specified on the **DTRTRAN** system initialization parameter, is to be rejected or accepted for processing.

This field is not used by the distributed routing program. On invocation, it is set to N.

DYRDTRXN
Indicates whether the transaction to be routed is defined by the common transaction definition specified on the **DTRTRAN** system initialization parameter or by a specific transaction definition.

This field is not used by the distributed routing program. On invocation, it is set to N.

DYRERROR

Has a value only when DYRFUNC is set to 1. It indicates the type of error that occurred during the last attempt at route selection. The possible values are:

0 The selected SYSID is unknown.

1 The selected system is not in service.

2 The selected system is in service, but no sessions are available.

3 An allocate request has been rejected, and SYSIDERR is returned to the application program. This error occurs for one of the following reasons:

- An XZIQUE global user exit program requested that the allocate be rejected
- CICS rejected the allocate request automatically because the QUEUELIMIT value specified on the CONNECTION resource definition was reached.

4 A queue of allocate requests has been purged, and SYSIDERR is returned to all the waiting application programs. This error occurs for one of the following reasons:

- An XZIQUE global user exit program requested that the queue be purged
- CICS purged the queue automatically because the MAXQTIME limit specified on the CONNECTION resource definition was reached.

5 The selected system does not support this function.

For BTS processes and activities and nonterminal-related START requests, this error occurs if the distributed routing program tries to route a request to a CICS region that is not connected by an MRO or APPC parallel-session link.

For inbound web services requests, this error occurs if the distributed routing program tries to route a request to a pre-CICS TS for z/OS, Version 3.1 region.

The next six values all apply to attempts to route START requests. For the meanings of these error conditions, see START in Reference -> Application development.

6 The **EXEC CICS START** command returned LENGERR.

8 The **EXEC CICS START** command returned INVREQ.

9 The **EXEC CICS START** command returned NOTAUTH.

C The **EXEC CICS START** command returned TRANSIDERR.

D The **EXEC CICS START** command returned IOERR.

E The **EXEC CICS STAR**T command returned USERIDERR.

F An XPCERES or XICERES global user exit program on the target region set a return code of UERCRESU, meaning that a required resource is unavailable on the target region. This error code can be set for program link, Link3270 bridge, and nonterminal-related START requests.

DYRFUNC
Tells you the reason for this invocation of the distributed routing program. These values are possible:

0 Invoked for route selection.

This invocation occurs on the routing region.

1 Invoked because an error occurred in route selection.

This invocation occurs on the routing region.

2 Invoked because the transaction associated with a previously routed request has ended successfully.

This invocation occurs on the target region.

3 Invoked for notification of the destination of a statically routed request.

This invocation occurs on the routing region. It applies in the following cases:

BTS processes and activities
A **RUN ASYNCHRONOUS** command has been issued, but the transaction associated with the BTS process or activity is defined as DYNAMIC(NO).

Inbound web service requests
The transaction associated with the request is defined as DYNAMIC(NO).

Nonterminal-related START requests
A transaction defined as ROUTABLE(YES) is eligible for *enhanced* routing but not for *dynamic* routing because one or both of the following applies:

- The transaction definition specifies DYNAMIC(NO).
- The SYSID option of the START command names a remote region explicitly.

For detailed information about which nonterminal-related START requests are eligible for dynamic routing, see Routing transactions invoked by START commands in Getting started.

4 Invoked because the transaction associated with the routed request abended.

This invocation occurs on the target region.

5 Invoked for transaction initiation. The transaction associated with the routed request is about to be started on the target region.

This invocation occurs on the target region.

6 Invoked because CICS has finished trying, successfully or unsuccessfully, to route the request to the target region.

This invocation occurs on the routing region. It signals that, unless the routing region and the target region are the same, the responsibility of the routing region for this transaction has been discharged. The routing program might, for example, use this invocation to release any resources that it has acquired on behalf of the transaction.

The DYRTYPE field tells you the type of routing or notification request.

DYRLEVEL
Is the level of CICS required in the target AOR to successfully process the routed request. These values are possible:

X'00' Any currently supported version of CICS is able to process the request.

X'01' CICS TS for z/OS, Version 2.2.

X'02' CICS TS for z/OS, Version 2.3.

X'03' CICS TS for z/OS, Version 3.1. This value is set for these requests:
- DPL requests that have a channel associated with them.

  Note: The routing of DPL requests is handled by the *dynamic* routing program.
- START requests that have a channel associated with them.
- Inbound web services requests.

X'04' CICS TS for z/OS, Version 3.2.

Note that values greater than X'00' indicate the *specific* not the minimum level of CICS required to process the request successfully.

DYRLPROG
Is not used by the distributed routing program. On invocation, it is set to null characters.

DYRNETNM
Is not used by the distributed routing program. On invocation, it is set to null characters. To set the target region, the distributed routing program must use the DYRSYSID field.

DYROPTER
Specifies whether the distributed routing program is to be reinvoked *on the target region* when the transaction associated with the routed request is to be started on the target region or ends (successfully or unsuccessfully).

This field is for use on route selection, notification, and route selection error calls. These values are possible:

N The distributed routing program is not to be invoked to start, to stop, or abend a transaction; that is, it is *not to be invoked on the target region*. N is the default.

Y The distributed routing program is to be reinvoked on the target region.

You can specify this option for requests that are routed to remote CICS regions and also for those that are executed locally.

DYRPROCCMP
Indicates whether the BTS process is completing.

This field applies only to the routing of BTS processes and activities. Its contents are significant on calls to stop a transaction.

These values are possible:

Y This call is the final activation of the BTS process.

N This call is not the final activation of the BTS process.

When the routing program is invoked when the transaction ends, the routing program can use the value of this field to decide whether to end any transaction affinities.

DYRPROCID
Is the CICS-assigned, 52-character identifier of the BTS process to which the activity being routed belongs.

This field applies only to the routing of BTS processes and activities.

DYRPROCN
Is the name of the BTS process to which the activity being routed belongs.

This field applies only to the routing of BTS processes and activities.

DYRPROCT
Is the process-type of the BTS process to which the process or activity being routed belongs.

This field applies only to the routing of BTS processes and activities.

DYRPRTY
Is not used by the distributed routing program. On invocation, it is set to zeros.

DYRQUEUE
Identifies whether the request is to be queued if no sessions are immediately available to the remote system identified by DYRSYSID.

This field is not used by the distributed routing program. On invocation, it is set to Y.

DYRRETC
Contains a return code that tells CICS how to proceed. These values are possible:

0 Route the request.

Non-zero
Do not route the request. CICS treats requests for BTS processes and activities as unserviceable. See the description of unserviceable requests in "If an error occurs in route selection" on page 613. START requests receive the SYSIDERR condition.

Whenever the routing program is invoked, DYRRETC is set to 0. When it is invoked for route selection or because an error occurs in route selection, if you want CICS to route the request to the region specified in the DYRSYSID field, you must leave it set to 0.

You do not have to set a return code when the routing program is invoked for notification, routing complete, starting or stopping a transaction, or an abend. Any code you set is ignored by CICS.

DYRRTPRI
Indicates whether the dispatch priority of the transaction is to be passed to the application-owning region, if the connection between the terminal-owning region and the application-owning region is MRO or IPIC.

This field is not used by the distributed routing program. On invocation, it is set to N.

DYRSRCTK
Is the MVS workload management service and reporting class token for the routed transaction. Your routing program must not alter this value, which is set by CICS and used by CICSPlex SM. For nonterminal-related START requests, this field is set to zeros. Do not change it.

DYRSYSID

Is the system identifier (SYSID) of a CICS region. The exact meaning of this parameter depends on the value of DYRFUNC:

- When DYRFUNC is set to 0 (route selection), DYRSYSID contains one of these names:
  - The CICS region name specified on the REMOTESYSTEM option of the installed transaction definition
  - If REMOTESYSTEM is not specified, the system name of the local CICS region.

  The distributed routing program can accept the value of DYRSYSID or change it before returning to CICS.

  If the SYSID you return to CICS is the same as the local SYSID, CICS runs the request on the local region.
- When DYRFUNC is set to 1 (route selection error), DYRSYSID contains the CICS region name returned to CICS by the distributed routing program on its previous invocation. If you want CICS to retry routing, you must change DYRSYSID before returning to CICS.
- When DYRFUNC is set to 2 (end of a routed request), DYRSYSID contains the name of the target region on which the completed transaction executed. This region is also the one on which the distributed routing program is invoked.
- When DYRFUNC is set to 3 (notification):
  - For BTS processes and activities, DYRSYSID contains one of these names:
    - The CICS region name specified on the REMOTESYSTEM option of the installed transaction definition.
    - If REMOTESYSTEM is not specified, the system name of the local CICS region.
  - For inbound web service requests, DYRSYSID contains one of these names:
    - The CICS region name specified on the REMOTESYSTEM option of the installed transaction definition (for the transaction named in the DFHWS-TRANID container).
    - If REMOTESYSTEM is not specified, the system name of the local CICS region.
  - For non-terminal-related START requests, DYRSYSID contains one of these names:
    - The remote CICS region name specified on the SYSID option of the **EXEC CICS START** command.
    - If SYSID is not specified, the remote CICS region name specified on the REMOTESYSTEM option of the installed transaction definition.
    - If REMOTESYSTEM is not specified, the system name of the local CICS region.

  Any change to the value of DYRSYSID is ignored.
- When DYRFUNC is set to 4 (transaction abend), DYRSYSID contains the name of the target region on which the transaction abended. This region is the one on which the distributed routing program is invoked.
- When DYRFUNC is set to 5 (transaction initiation), DYRSYSID contains the name of the target region on which the routed request is to be executed. This region is the one on which the distributed routing program is invoked.

- When DYRFUNC is set to 6 (routing completed), DYRSYSID contains the name of the target region to which CICS tried (successfully or unsuccessfully) to route the request.

DYRTRAN
Contains the transaction name.

Note that this is *the name by which the transaction is known in the routing region*. Unlike the dynamic routing program, the distributed routing program is passed the local, not the remote, transaction name and cannot specify an alternative remote transaction name, for forwarding to the target region.

DYRTYPE
Is the type of routing request for which the program is being invoked. The values that can be passed to the *distributed* routing program are as follows:

5 A BTS process or activity.

6 A nonterminal-related START request, with or without data but with no channel.

7 A method request for an inbound web services request.

B A nonterminal-related START request, with a channel.

DYRUAPTR
If DYRVER is 7 or greater, this field contains the address of the new user area, DYRUSERN. The new user area mechanism makes the source of the routing program independent of the CICS release that created the communications area. The old user area field DYRUSER is retained only for compatibility purposes.

The user area can be mapped with the DYRUAREA DSECT.

To ensure that DYRVER is 7 or greater, you must apply the PTFs for the following APARs to any of your routing or target regions tup to and including CICS TS for z/OS, Version 2.3:

CICS Transaction Server for OS/390, Version 1 Release 3
PQ75814

CICS Transaction Server for z/OS Version 2 Release 2
PQ75834

CICS Transaction Server for z/OS Version 2 Release 3
PQ81378

In systems where DYRUAPTR is less than 7, the contents of DYRUAPTR are unpredictable.

DYRUSER
Is a 1024-byte user area.

This field is retained only for compatibility purposes; see the descriptions of the DYRUAPTR and DYRUSERN fields.

DYRUSERID
Is the CICS user ID associated with the request.

- For BTS processes and activities, DYRUSERID contains one of these user IDs:
  - If the BTS process or activity was activated by a LINK ACQPROCESS or LINK ACTIVITY command, the user ID of the transaction that issued the LINK.
  - The user ID specified on the USERID option of the DEFINE PROCESS or DEFINE ACTIVITY command.

– If USERID was not specified on the DEFINE command, the user ID under which the transaction that issued the DEFINE command is running.

For further details of the user IDs associated with BTS processes and activities, see the *CICS Business Transaction Services* manual.

- For inbound web services requests, DYRUSERID contains the user ID associated with the request stream.
- For nonterminal-related START requests, DYRUSERID contains:
  – The user ID specified on the USERID option of the **EXEC CICS START** command.
  – If USERID is not specified, the user ID under which the transaction that issued the START command is running.

By examining this field when it is invoked for routing or because of a route-selection error (DYRFUNC=0 or 1, respectively), your routing program can route requests based on the user ID associated with the request.

DYRUSERN
Is a 1024-byte user area.

CICS initializes this user area to zeros before invoking the distributed routing program for a given task. This user area can be modified by the routing program; the modified area is passed to subsequent invocations of the routing program for the same request.

1. The user area passed to the routing program on its first call on the target region (transaction initiation) is a *copy* of the user area on the routing region. Therefore, any modifications to the user area made on the target region have no effect on the user area in the routing region. For example, a change to the user area made on the call to start the transaction has no effect on the user area passed to the routing complete call, even if the latter occurs after the call to start the transaction.
2. The user area passed to the first (transaction start) call on the target region is a copy of that returned by *the call on the routing region that caused the transaction start call to occur*. That is:
   - If the route selection contained no error, it is a copy of the user area returned by the route selection or notification call.
   - If a route selection error occurred, it is a copy of the user area returned by the final route selection error call.
   - It is *never* a copy of the user area returned by the routing attempt complete call on the routing region, even if the latter occurs before the transaction start call on the target region.

DYRVER
Is the version number of the dynamic routing interface. For CICS Transaction Server for z/OS, Version 5 Release 2, the number is 10.

Naming your distributed routing program

The supplied, sample distributed routing program is named DFHDSRP. If you write your own version of this program you can name it differently.

About this task

After the system has been loaded, use the CICS Explorer **Regions** operations view or the **EXEC CICS INQUIRE SYSTEM** command to find the name of the distributed

routing program currently identified to CICS. The field DSRTPROGRAM contains the name of the current program.

Procedure

To change the current program, you can use one of the following methods:

1. Use the CICS Explorer **Regions** operations view.
2. Use the DSRTPGM system initialization parameter.
3. Use the **EXEC CICS SET SYSTEM DSRTPROGRAM** command. For programming information about this command, see SET SYSTEM in Reference > System programming.

A sample definition is provided for DFHDSRP, but you must install a new resource definition for a customized distributed routing program.

Distributed transaction routing sample programs

The CICS-supplied sample distributed routing program is named DFHDSRP. The corresponding copy book that defines the communications area is DFHDYPDS.

There are assembler-language, COBOL, PL/I, and C source-level samples and copy books. The supplied programs and copy books, and the CICSTS52.CICS libraries in which they can be found, are summarized in the following tables.

Table 33. Distributed routing programs

Language	Member name	Library
Assembler	DFHDSRP	SDFHSAMP
COBOL	DFHDSRP	SDFHCOB
PL/I	DFHDSRP	SDFHPL1
C	DFHDSRP	SDFHC370

Table 34. Copy books

Language	Member name	Library
Assembler	DFHDYPDS	SDFHMAC
COBOL	DFHDYPDS	SDFHCOB
PL/I	DFHDYPDS	SDFHPL1
C	DFHDYPDS	SDFHC370

You can write your own distributed routing program in COBOL, PL/I, C, or assembler language, and you can change the name of the program.

When invoked with DYRFUNC set to 0, the sample programs accept the SYSID that is passed in field DYRSYSID of the communications area, and set DYRRETC to 0 before returning to CICS. When invoked with DYRFUNC set to 2, 3, 5, or 6, they set a return code of 0. When invoked with DYRFUNC set to 1 or 4, they set a return code of 8.

If you want to route requests dynamically, you must customize DFHDSRP, or replace it completely with your own routing program.

Chapter 21. Writing a CICS–DBCTL interface status program

The CICS–DBCTL interface status program DFHDBUEX is a user-replaceable program forming part of the support for the CICS–DBCTL interface. It is designed to invoke user-supplied code whenever CICS successfully connects to or disconnects from DBCTL. It runs in a CICS application environment and is driven at specific points to allow you to enable and disable your CICS-DL/I transactions when the CICS–DBCTL interface initializes or terminates.

DFHDBUEX is called in the following case for the ENABLE command:

- CICS has connected to DBCTL successfully. This occurs after a connection request has been issued from CICS to DBCTL. The control exit (DFHDBCTX) is invoked by the database resource adapter (DRA) for 'initialization complete'. The control exit posts the control transaction (CDBO). The control program (DFHDBCT) then invokes DFHDBUEX.

DFHDBUEX is called in the following cases for the DISABLE command:

- A request has been issued to disconnect from DBCTL. The CICS–DBCTL menu program (DFHDBME) starts the disconnection transaction (CDBT) to disconnect from DBCTL. The disconnection program (DFHDBDSC) invokes DFHDBUEX before issuing the interface termination request to the adapter.
- The control transaction (CDBO) has been notified of one of the following events:
  - A checkpoint freeze request to DBCTL
  - DRA abnormal termination
  - DBCTL abnormal termination.

  In each of these cases, the control program (DFHDBCT) invokes DFHDBUEX.

Input to DFHDBUEX is by means of a communication area addressed by DFHEICAP. The layout of the communication area is shown in Figure 72.

```
DBUSHEAD    DS      OCL4            Standard Header
DBUREQT     DS      CL1             Function Code
DBUCOMP     DS      CL2             Component Code         Always "DB"
DBURESV     DS      CL1             Reserved
DBUREAS     DS      CL1             Reason for disconnection
DBUSUFF     DS      CL2             DRA startup table suffix
DBUDBCTL    DS      CL4             DBCTL identifier
```

Figure 72. The DFHDBUEX communication area

The parameter list contains the following information:

DBUREQT
Request Type. The function code has one of the following values:

DBUCONN (X'01')
Connected

DBUDISC (X'02')
Disconnected.

DBUREAS
Reason for Disconnection. Contains flags:

DBUMENU (X'01')
Disconnected from menu

DBUDBCC (X'02')
Checkpoint Freeze input to DBCTL

DBUDRAF (X'03')
DRA Failure has taken place

DBUDBCF (X'04')
DBCTL Failure has taken place.

DBUSUFF
DRA startup table suffix.

DBUDBCTL
DBCTL identifier.

The sample CICS–DBCTL interface status program

The source-code of the supplied CICS–DBCTL interface status program, DFHDBUEX, is provided, in assembler language only, in the CICSTS52.CICS.SDFHSAMP library. A corresponding copy book, DFHDBUCA, that maps the communication area, is in CICSTS52.CICS.SDFHMAC.

The sample program checks for the presence of the input parameters (passed in the communication area). If these do not exist, control returns to the calling program.

The type of request (CONNECTION | DISCONNECTION) is then determined, and a branch is taken to the appropriate function routine (CONPROC | DISPROC).

The sample contains an example, as part of a comment, of how to enable and how to disable a transaction. To use the program, it is necessary for transactions using DBCTL to be defined in the CSD as DISABLED.

You can code your own CICS–DBCTL interface status program in any of the languages supported by CICS. For information about the job control statements necessary to assemble and link-edit user-replaceable programs, refer to Chapter 6, “Assembling and link-editing user-replaceable programs,” on page 419.

Chapter 22. Writing a 3270 bridge exit program

The 3270 bridge provides an interface so that you can run 3270-based CICS transactions without a 3270 terminal. You can write a program that receives intercepted terminal commands in the bridge environment.

The bridge exit provides the mechanism by which data needed to run a 3270 transaction can be sent and received from an external resource. For example, you can use WebSphere MQ commands in a bridge exit so that CICS can get and put messages on WebSphere MQ queues.

For a detailed description of the 3270 bridge exit and its interfaces, see Introduction to the 3270 bridge in Developing applications.

Chapter 23. Writing programs to customize Language Environment runtime options for XPLink programs

User-replacable program DFHAPXPO is called by CICS during the initialization of the Language Environment enclave for a C or C++ program compiled with the XPLINK option.

DFHAPXPO

This program is loaded during the PIPI preinitialization phase of each Language Environment enclave where C or C++ programs compiled with the XPLINK option are to be run. It allows you to alter the default Language Environment runtime options. See the *z/OS Language Environment Programming Guide* for details of the Language Environment options that can be reset. The program must be written in Assembler language.

Defining run-time options

The options are specified as a human-readable string containing a 2-byte string length followed by the run-time options.

The maximum length allowed for all Language Environment run-time options is 255 bytes, so you are recommended to use the abbreviated version of each option and restrict your changes to a total of under 200 bytes. The values you specify are not checked by CICS before being passed to Language Environment.

A CICS-supplied DFHAPXPO module is provided, setting some run-time options. See the fully-commented module source for an example of how to set these options.

CICS programs can include a CEEUOPTS CSECT to supply Language Environment run-time options to control the program's execution.

Chapter 24. The user-replaceable conversion program

This section describes the user-replaceable data conversion program.

User-named conversion programs

You can replace DFHUCNV, the default user-replaceable conversion program, by one or more user-named conversion programs.

DFHUCNV is invoked if:

- A conversion template is not defined for the resource, *or*
- A conversion template is defined for the resource and the template specifies `USREXIT=YES`.

A user-named conversion program is invoked if:

- A conversion template is defined for the resource and the template specifies `USREXIT=`*`userprogram`*

  where *userprogram* is the name of the user-supplied conversion program.

Input to DFHUCNV

The first statement in the supplied version of DFHUCNV is a DFHCNV TYPE=DSECT macro, which generates DSECTs that describe the parameter list and the conversion template.

DFHUCNV starts with a DFHCNV TYPE=DSECT in the following format:

```
DFHCNV  TYPE=DSECT
```

The DFHCNV TYPE=DSECT macro generates the following:

- The DFHUNVDS DSECT, which maps the parameter list in the COMMAREA passed by DFHCCNV.
- An assembler DSECT for field conversion records (these are the basic components of a template; see Figure 75 on page 643).
- Equates for resource types and field types.

Parameter list (DFHUVNDS)

The DFHUNVDS DSECT maps the parameter list passed to DFHUCNV in the COMMAREA.

If a parameter is zero, no data is available. *If you do not create a conversion template for the resource, DFHUCNV is invoked, but only the following fields in the parameter list contain data*:

- UNVRSTP
- UNVRNMP
- UNVDIRP
- UNVOVLY

```
DFHUNVDS DSECT
UNVRSTP  DS    AL4                PTR-TO-RESOURCE TYPE
UNVRNMP  DS    AL4                PTR-TO-RESOURCE NAME
UNVDIRP  DS    AL4                PTR-TO-CONVERSION DIRECTIVE
CNVRQATE EQU   X'02'               REQUEST ASCII TO EBCDIC
CNVRPETA EQU   X'04'               RESPONSE EBCDIC TO ASCII
UNVDTMP  DS    AL4                PTR-TO-DATA CONV TEMPLATE
UNVDLNP  DS    AL4                PTR-TO-DATA TEMPLATE LENGTH
UNVKTMP  DS    AL4                PTR-TO-KEY CONV TEMPLATE
UNVKLNP  DS    AL4                PTR-TO-KEY TEMPLATE LENGTH
UNVATEP  DS    AL4                PTR-TO-ASCII/EBCDIC TRANS TABLE
UNVETAP  DS    AL4                PTR-TO-EBCDIC/ASCII TRANS TABLE
UNVATED  DS    AL4                PTR-TO-DBCS ASCII/EBCDIC TRANS TABLE
UNVETAD  DS    AL4                PTR-TO-DBCS EBCDIC/ASCII TRANS TABLE

UNVOVLY  DS    0H                 OVERLAY SECTION
         ORG   UNVOVLY            TS REQUEST OVERLAY
UNVTSDP  DS    AL4                PTR-TO-TS DATA
UNVTSLNP DS    AL4                PTR-TO-TS DATA LENGTH
         ORG   UNVOVLY            TD REQUEST OVERLAY
UNVTDDP  DS    AL4                PTR-TO-TD DATA
UNVTDLNP DS    AL4                PTR-TO-TD DATA LENGTH
         ORG   UNVOVLY            IC REQUEST OVERLAY
UNVICDP  DS    AL4                PTR-TO-IC DATA
UNVICLNP DS    AL4                PTR-TO-IC DATA LENGTH
         ORG   UNVOVLY            PC REQUEST OVERLAY
UNVPCDP  DS    AL4                PTR-TO-PC DATA
UNVPCLNP DS    AL4                PTR-TO-PC DATA LENGTH
         ORG   UNVOVLY            FC REQUEST OVERLAY
UNVFCDP  DS    AL4                PTR-TO-FC DATA
UNVFCLNP DS    AL4                PTR-TO-FC DATA LENGTH
UNVFCKP  DS    AL4                PTR-TO-FC KEY
UNVFCKLP DS    AL4                PTR-TO-FC KEY  LENGTH
         ORG   ,
UNVMRTNE DS    A                  PTR-TO-MBCS TRANSLATION ROUTINE
UNVCLIDP DS    AL4                A "client" CCSID
*                                   (for example, 00819)
UNVSRIDP DS    AL4                A "server" CCSID
*                                   (for example, 00285)
```

Figure 73. DFHUNVDS—DSECT that maps the parameter list passed to DFHUCNV

The following is a detailed description of the parameters:

UNVRSTP
Points to a one-byte resource type that indicates the resource being referenced by this request. The meanings of the resource types are defined in DSECT DFHCNVDS. The resource types are FC, IC, TS, TD, and PC.

UNVRNMP
Points to an eight-character field containing the resource name, padded with blanks if necessary. These may be:

- For an FC request, an eight-byte file name
- For a TS request, an eight-byte TS queue name
- For a TD request, a four-byte TD queue name
- For an IC request, a four-byte transaction name
- For a PC request, an eight-byte program name.

UNVDIRP
Points to a one-byte field that shows what conversion is required:

- CNVRQATE (X'02') indicates a request needing conversion from client encoding to server encoding.

- CNVRPETA (X'04') indicates a response needing conversion from server encoding to client encoding.

UNVDTMP
Points to the start of the conversion template found by CICS to match this resource. If UNVDTMP is zero no template was found.

UNVDLNP
Points to a field that gives the length of the conversion template. The field is:
- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

UNVKTMP (file control requests only)
Points to the start of the template found by CICS for the key part of the request or response. If UNVKTMP is zero, either there is no key template or the record is accessed by relative record number or relative byte address.

UNVKLNP (file control requests only)
Points to a field that gives the length of the key conversion template. The field is:
- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

UNVATEP
Points to a 256-byte SBCS translation table used for converting character data from client encoding to server encoding.

UNVETAP
Points to a 256-byte SBCS translation table used for converting character data from server encoding to client encoding.

UNVATED
Points to a DBCS translation table used for converting character data from client encoding to server encoding.

UNVETAD
Points to a DBCS translation table used for converting character data from server encoding to client encoding.

The overlay section depends on resource type:

TS requests:

UNVTSDP
Points to the start of the TS record being read or written. The field is:
- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

UNVTSLNP
Points to a field that gives the length of the TS record.

TD requests:

UNVTDDP
Points to the start of the TD record being read or written.

UNVTDLNP
Points to a field that gives the length of the TD record. The field is:
- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

IC requests:

UNVICDP
Points to the "from" area of an IC START request.

UNVICLNP
Points to a field that gives the length of the "from" area. The field is:
- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

PC requests:

UNVPCDP
Points to the start of the COMMAREA being supplied.

UNVPCLNP
Points to a field that gives the length of the COMMAREA. The field is:
- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

FC requests:

UNVFCDP
Points to the start of the file control record being read or written.

UNVFCLNP
Points to a field that gives the length of the file control record. The field is:
- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

UNVFCKP
Points to the start of the key for the file control record being read or written.

UNVFCKLP
Points to a field that gives the length of the key. The field is:
- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

UNVMRTNE
Points to a translation routine that must be used for translations to or from an MBCS code page. The relevant client code pages are 954, 964, and 970.

The routine expects Register 1 to point to a structure defined by the DFHUNVM DSECT:

```
DFHUNVM DSECT
UNVMTABP  DS    AL4           Set to value in UNVATED or UNVETAD
UNVMINP   DS    AL4           Address of source data
INVMINL   DS    FL4           Length of source data
UNVMOUTP  DS    AL4           Address of target buffer
UNVMOUTL  DS    FL4           Length of target buffer
```

UNVCLIDP
Points to a fullword field that gives the IBM-defined CCSID, for example 00819, corresponding to the "client" code page.

UNVSRIDP
Points to a fullword field that gives the IBM-defined CCSID, for example 00285, corresponding to the "server" code page.

Conversion and key templates

In the COMMAREA, fields UNVDTMP and UNVDLNP point to the conversion template and its length.

Fields UNVKTMP and UNVKLNP point to the key template and its length. Figure 74 illustrates the use and meaning of these fields.

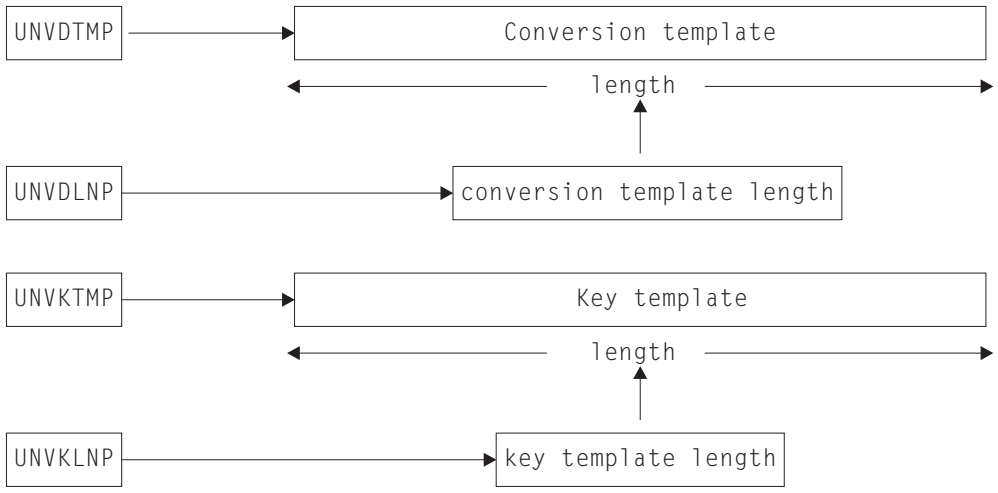


Figure 74. Parameter fields and the conversion templates

Each type of template consists of field conversion records, one for each field in the data record or key. Each field conversion record has the same layout, shown under "Field conversion records," and mapped by a supplied DSECT, DFHCNVDS (see "DFHCNVDS, DSECT for field conversion records" on page 644). Figure 75 shows the relationship between a template, field conversion records, and DFHCNVDS. The figure shows DFHCNVDS overlaying the first field conversion record in a template for a data record or key with six fields.

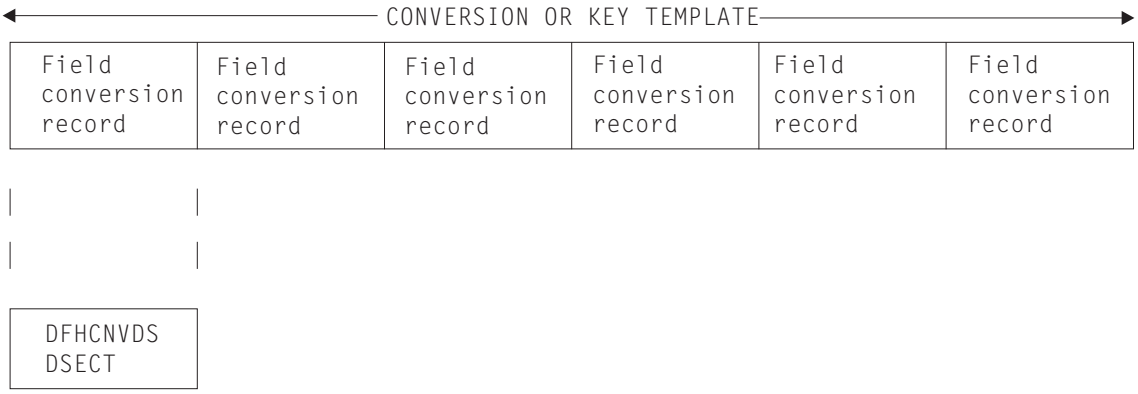


Figure 75. Field conversion records and a conversion or key template

Field conversion records

The layout of the field conversion records are described.

A field conversion record has the following layout:

Table 35. Layout of a field conversion record

CNVRLEN	CNVRTYPE	Reserved	CNVDATTY	CNVDATAO	CNVDATAL
Record length	Record type	Reserved	Data type	Data offset	Data length

Table 35. Layout of a field conversion record (continued)

CNVRLEN	CNVRTYPE	Reserved	CNVDATTY	CNVDATAO	CNVDATAL
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5-8	Byte 9-12

In Table 35 on page 643, record length and type refer to the length and type of the field conversion record. The names in the first row are those used in the DSECT DFHCNVDS, which maps field conversion records (see "DFHCNVDS, DSECT for field conversion records"). A template has as many field conversion records as are necessary to describe all the fields in the data record or key.

For DFHUCNV, CNVRLEN is X'0C' and CNVRTYPE is always X'04' (field). DFHUCNV must interpret CNVDATTY values in the range X'50' through X'80' according to user specifications, and apply the appropriate conversions. DFHUCNV should ignore fields with CNVDATTY values outside the range X'50' to X'80'.

EQUATEs in DFHCNVDS

DFHCNVDS contains EQUATEs that are useful in your conversion program.

For resource type addressed by the parameter list

```
CNVFC           FILE CONTROL
CNVTS           TEMPORARY STORAGE
CNVTD           TRANSIENT DATA
CNVIC           INTERVAL CONTROL
CNVPC           PROGRAM CONTROL
```

For field type in the template

Two additional EQUATEs, DTUSRMIN and DTUSRMAX, define the limits of the range of data types (X'50' to X'80') reserved for user definition. Ensure that DFHUCNV can deal with any data type in this range that can be used in your installation.

```
DTBIN           BINARY
DTPD            PACKED DECIMAL
DTCHAR          CHARACTER
DTMIX           MIXED CHARACTER
DTDBCS          DBCS CHARACTER
DTNUM           INTEL INTEGER
```

The supplied DFHUCNV program contains examples of the use of CNVTS, DTUSRMIN, and DTUSRMAX—see "Supplied user-replaceable conversion program" on page 645.

DFHCNVDS, DSECT for field conversion records

```
DFHCNVDS DSECT
*
*        PROVIDES A MAPPING OF THE FIELD CONVERSION RECORDS USED
*        WHEN DECIDING WHETHER TO CONVERT USER DATA.
*        A SET OF FIELD DEFINITIONS MAKE UP A TEMPLATE
*
CNVRLEN  DS    AL1                LENGTH OF THIS RECORD
CNVRTYPE DS    XL1                TYPE OF RECORD
*
*        EQUATES FOR RECORD TYPES
*
CNVTFLD  EQU   X'04'               FIELD (ONLY VALID TYPE IN
*                                  TEMPLATE)
CNVOVLY  DS    0H
**
**
         ORG   CNVOVLY             TYPE FIELD
         DS    XL1                 RESERVED
CNVDATTY DS    XL1                 DATA TYPE
*
*        EQUATES FOR DATA TYPES
*
DTBIN    EQU   X'01'               BINARY
DTPD     EQU   X'02'               PACKED DECIMAL
DTCHAR   EQU   X'03'               CHARACTER
DTMIX    EQU   X'04'               MIXED CHARACTER
DTDBCS   EQU   X'05'               DBCS
DTNUM    EQU   X'06'               NUMERIC
DTUSRMIN EQU   X'50'               MINIMUM USER DATA TYPE
DTUSRMAX EQU   X'80'               MAXIMUM USER DATA TYPE
*
CNVDATAO DS    AL4                 DATA OFFSET
CNVDATAL DS    AL4                 DATA LENGTH
**
*
*        EQUATES FOR RESOURCE TYPES
*
CNVFC    EQU   X'01'               FILE CONTROL
CNVTS    EQU   X'02'               TEMP STORAGE
CNVTD    EQU   X'03'               TRANS DATA
CNVIC    EQU   X'05'               INTERVAL CONTROL
CNVPC    EQU   X'06'               PROGRAM CONTROL
```

Figure 76. DFHCNVDS, DSECT that maps conversion/key templates passed to DFHUCNV

Supplied user-replaceable conversion program

The supplied version of DFHUCNV checks for a resource type of TS. If it finds one, it scans down the passed template looking for fields defined with a type in the user-data range. If any are present, DFHUCNV converts them as characters; you can rewrite the conversion code to your own requirements.

Study the supplied version of DFHUCNV and its introductory comments to enable you to write your own conversion program. Your program must be able to handle 31 bit addresses.

The supplied sample is defined to CICS with program attribute CONCURRENCY(THREADSAFE). Any code added to the sample must be threadsafe as the program might be started on an open TCB. Alternatively you can change the program definition to specify CONCURRENCY(QUASIRENT) but this change can produce a TCB switching overhead.

Part 4. Writing statistics collection and analysis programs

You can customize the collection and analysis of CICS statistics by writing your own statistics collection and analysis programs.

When writing a statistics collection and analysis program, you must ensure that CICS always receives control in primary-space translation mode. The original contents of all access registers must be restored, and all general purpose registers must be restored (except for those which provide return codes or linkage information). For information about translation modes, see *IBM z/Architecture Principles of Operation*.

Chapter 25. Writing a program to collect CICS statistics

Why collect CICS statistics?

CICS statistics contain information about the CICS system as a whole; for example, its performance and usage of resources. Statistics data is, therefore, useful both for performance tuning and for capacity planning.

Statistics are collected during CICS online processing for later offline analysis. The statistics domain writes statistics records to a System Management Facility (SMF) data set. The records are of SMF type 110, subtype 0002.

Note: Monitoring records, and statistics records produced by the temporary storage shared-queue server, are also written to the SMF data set as type 110 records. (Some journaling type 110 records can be written there, too.) You might find it useful to process the statistics records and the monitoring records together, because statistics provide resource and system information that is complementary to the transaction data produced by CICS monitoring.

Statistics records are also written by:

- Temporary storage (TS) data sharing pool server regions. These records are of SMF type 110, subtype 0003.
- Coupling facility data table (CFDT) server regions. These records are of SMF type 110, subtype 0004.
- Named counter sequence number server regions. These records are of SMF type 110, subtype 0005.

CICS produces five types of statistics: **interval, end-of-day, requested, requested reset**, and **unsolicited**.

Important

For detailed information about the types of CICS statistics, when they are collected, and how to control their collection, see Introduction to CICS statistics, in the *CICS Performance Guide*.

Reset options for statistics counters

Statistics counters are reset in the following circumstances:

- At CICS startup
- When interval statistics are written (but not when an interval occurs and no statistics are written)
- At end of day
- When requested reset statistics are written

However, you can cause statistics counters to be reset without writing records to the SMF data set. Change the statistics recording status, by using one of these options:

- The **Reset Time** option of the CICS Explorer **Regions** operations view
- The **CEMT SET STATISTICS ON|OFF RESETNOW** command

- The **EXEC CICS SET STATISTICS ON|OFF RESETNOW** command

In this way, a user program can reset all statistics counters.

It is valid to specify the RESETNOW option only when there is a genuine change of recording status. For example, coding **EXEC CICS SET STATISTICS ON RESETNOW** when STATISTICS is already set ON causes an error response.

Important

Statistics counters are reset in various ways. Specific counters can be reset to:

- 0
- 1
- A new peak value
- Not reset
- None of the above

For information about the resetting of specific statistics counters, see , in the *CICS Performance Guide*.

Collecting and extracting CICS statistics

The statistics collected by CICS are written to an SMF data set. However, a user program can use the CICS Explorer **Regions** operations view, the **EXEC CICS COLLECT STATISTICS** command and the **EXEC CICS EXTRACT STATISTICS** command to collect the current statistics for a particular resource, or overall statistics for the resources of a particular type.

About this task

The kinds of statistics that you can collect might include statistics for global transaction activity in your CICS region, such as the total number of transactions attached. Alternatively you could specify a single transaction that you are interested in. The statistics are returned to the starting application. For programming information about these commands, see COLLECT STATISTICS.

Procedure

Perform your statistical analysis. CICS supplies 15 sample programs that show how you can use the **EXEC CICS COLLECT STATISTICS**, **EXEC CICS EXTRACT STATISTICS**, and **EXEC CICS INQUIRE** commands to produce a useful analysis of a CICS system. These are the programs:

- DFH0STAT
- DFH0STDB
- DFH0STEJ
- DFH0STEP
- DFH0STGN
- DFH0STLK
- DFH0STPR
- DFH0STSA
- DFH0STSY
- DFH0STTP
- DFH0STTS
- DFH0STWB
- DFH$STAS

- DFH$STCN
- DFH$STTB

The sample programs produce a report showing critical system parameters from the CICS dispatcher, together with loader statistics and an analysis of the CICS storage manager. DFH$STAS, DFH$STCN, and DFH$STTB are provided in assembler language; the other 12 programs are provided in COBOL.
For information about installing and operating the sample statistics programs, and about the data produced by the programs, see CICS statistics in Monitoring.

Chapter 26. CICS statistics record format

This section describes the format of CICS statistics SMF type 110 records in detail. You need this information if you write your own program to analyze the statistics data. The three components of a CICS statistics record are an SMF header, an SMF product section, and a CICS data section, as shown in Figure 77. Each of these is described in the sections that follow.

Figure 77. Format of an SMF type 110 statistics record

SMF header and SMF product section

The SMF header describes the system creating the output. The SMF product section identifies the subsystem to which the statistics data relates, which, in the case of CICS statistics, is the CICS region, the TS data sharing server, the CFDT server, or the named counter sequence number server.

Both the SMF header and the SMF product section can be mapped by the DSECT STSMFDS, which you can generate using the DFHSTSMF macro as follows:

```
STSMFDS DFHSTSMF PREFIX=SMF
```

The label 'STSMFDS' is the default DSECT name, and SMF is the default PREFIX value, so you could also generate the DSECT by coding `DFHSTSMF`.

The STSMFDS DSECT has the format shown in Figure 78 on page 654.

```
*        START THE SMF HEADER
*
STSMFDS  DSECT
SMFSTLEN DS    XL2           RECORD LENGTH
SMFSTSEQ DS    XL2           SEGMENT DESCRIPTOR
SMFSTFLG DS    X             OPERATING SYSTEM INDICATOR (see note 1)
SMFSTRTY DC    X'6E'         RECORD TYPE 110 FOR CICS
SMFSTTME DS    XL4           TIME RECORD MOVED TO SMF
SMFSTDTE DS    XL4           DATE RECORD MOVED TO SMF
SMFSTSID DS    XL4           SYSTEM IDENTIFICATION
SMFSTSSI DS    CL4'CICS'     SUBSYSTEM IDENTIFICATION
SMFSTSTY DS    XL2           RECORD SUBTYPE X'0002' FOR STATISTICS
*                                          (see note 4)
SMFSTTRN DS    XL2           NUMBER OF TRIPLETS
         DS    XL2           RESERVED
SMFSTAPS DS    XL4           OFFSET TO PRODUCT SECTION
SMFSTLPS DS    XL2           LENGTH OF PRODUCT SECTION
SMFSTNPS DS    XL2           NUMBER OF PRODUCT SECTIONS
SMFSTASS DS    XL4           OFFSET TO DATA SECTION
SMFSTASL DS    XL2           LENGTH OF DATA SECTION
SMFSTASN DS    XL2           NUMBER OF DATA SECTIONS
*
*        THIS CONCLUDES THE SMF HEADER
*
*        START THE SMF PRODUCT SECTION
*
SMFSTRVN DS    XL2           RECORD VERSION
SMFSTPRN DS    CL8           PRODUCT NAME (GENERIC APPLID)
SMFSTSPN DS    CL8           PRODUCT NAME (SPECIFIC APPLID)
SMFSTMFL DS    XL2           RECORD MAINTENANCE INDICATOR
         DS    XL2           RESERVED
         DS    XL2           RESERVED
SMFSTDTK DS    XL4           DOMAIN TOKEN
SMFSTDID DS    CL2           DOMAIN ID
SMFSTRQT DS    CL3           USS/EOD/REQ/INT STATISTICS TYPE
SMFSTICD DS    CL3           YES IF INCOMPLETE DATA RECORDED
SMFSTDAT DS    CL8           COLLECTION DATE MMDDYYYY
SMFSTCLT DS    CL6           COLLECTION TIME HHMMSS
SMFSTINT DS    CL6           INTERVAL HHMMSS. See note 3.
SMFSTINO DS    XL4           INTERVAL NUMBER. See note 3.
SMFSTRTK DS    XL8           REQUEST TOKEN
SMFSTLRT DS    CL6           LAST RESET TIME HHMMSS
SMFSTCST DS    XL8           CICS START TIME
SMFSTJBN DS    CL8           JOBNAME
SMFSTRSD DS    XL4           JOB DATE
SMFSTRST DS    XL4           JOB TIME
SMFSTUIF DS    CL8           USER IDENTIFICATION
SMFSTPDN DS    CL8           OPERATING SYSTEM PRODUCT LEVEL
*
*        THIS CONCLUDES THE SMF PRODUCT SECTION
```

Figure 78. Format of the SMF header and product section for statistics records

Note:

1. CICS sets only the subsystem-related bits of the operating system indicator flag byte in the SMF header (SMFSTFLG). SMF sets the remainder of the byte according to the operating system level and other factors. For an explanation of the setting of the other bits, refer to the *z/OS MVS System Management Facilities (SMF)* manual.
2. The copy book DFHSMFDS is also provided and can be used to map the SMF header and the SMF product sections of all six subtypes of SMF 110 records written by CICS journaling, CICS monitoring, and CICS statistics.
3. Fields SMFSTINT and SMFSTINO are only relevant if SMFSTRQT is 'INT'. Otherwise both values should be ignored.

4. For TS data sharing, the record subtype is X'0003' and certain fields are not set or are used in a different way. SMFSTPRN and SMFSTSPN contain the server prefix (DFHXQ) and the pool name.
5. For coupling facility data table (CFDT) servers, the record subtype is X'0004' and certain fields are not set or are used in a different way. SMFSTPRN and SMFSTSPN contain the server prefix (DFHCF) and the coupling facility data table pool name.
6. For named counter sequence number servers, the record subtype is X'0005' and certain fields are not set or are used in a different way. SMFSTPRN and SMFSTSPN contain the server prefix (DFHNC) and the pool name.

CICS statistics data section

The format of the CICS statistics data section is shown in Figure 79.
If the data records are incomplete, the flag field SMFSTICD is set to YES. In this

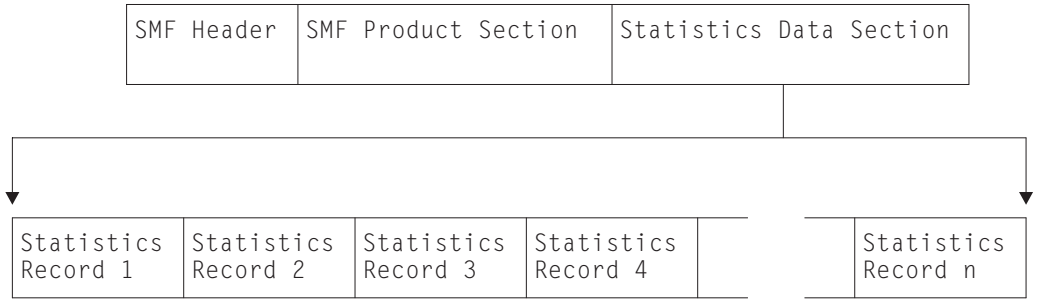


Figure 79. Format of the statistics data section

case, the statistics data section is not present.

For complete data records, the statistics data section is made up of one or more statistics data records. There are different formats of data records. Each has a common format for the first 5 bytes. These 5 bytes are described in the extract from copybook DFHSTIDS in Figure 80.

```
DFHSTIDS DSECT                   Statistics record header
*
         DS    0F                Fullword alignment
STILEN   DS    H                 Length of the record
STID     DS    AL2               Statistics identifier
STIVERS  DS    CL1               Statistics record version
```

Figure 80. Extract from copybook DFHSTIDS

STILEN
The length of the data record.

STID
A name or value that identifies which type of statistics record you have (see Figure 81 on page 657).

You can use the STID symbolic name or value to determine which copy book to use when processing the statistics data records. For details about the relationship between the STID name or value and the copy book, see Figure 81 on page 657. For further guidance information about the fields within the statistics data records, see CICS statistics in DSECTs and DFHSTUP report, in the *CICS Performance Guide*.

STIVERS
The version of the record. STIVERS takes the value 1 for this release of CICS.

STID Symbolic name	STID Value	Copybook	Type of record
STIXMG	10	DFHXMGDS	Transaction manager (Globals) id
STIXMR	11	DFHXMRDS	Transaction manager (Trans) id
STIXMC	12	DFHXMCDS	Transaction manager (Tclass) id
STIFEPIP	16	DFHA22DS	FEPI pool id
STIFEPIC	17	DFHA23DS	FEPI connection id
STIFEPIT	18	DFHA24DS	FEPI target id
STISMD	19	DFHSMDDS	Storage mgr domain subpool id
STISMT	20	DFHSMTDS	Storage manager task subpool id
STIVT	21	DFHA03DS	z/OS Communications Server stats id
STIPAUTO	23	DFHPGGDS	Program Autoinstall id
STIAUTO	24	DFHA04DS	Terminal Autoinstall stats id
STILDR	25	DFHLDRDS	Public Loader (Resid) id
STIDBUSS	28	DFHDBUDS	DBCTL USS id
STISMDSA	29	DFHSMSDS	Storage manager DSA id
STILDG	30	DFHLDGDS	Loader (Globals) id
STILDB	31	DFHLDBDS	LIBRARY resources - public
STILDY	32	DFHLDYDS	LIBRARY resources - private
STITCR	34	DFHA06DS	Terminal control (resid) id
STILDP	36	DFHLDPDS	Private Loader (Resid) id
STILSRR	39	DFHA08DS	LSRPOOL pool stats (resid) id
STILSRFR	40	DFHA09DS	LSRPOOL File statistics (by file)
STITDQR	42	DFHTQRDS	TDQUEUE (Resid) id
STITDQG	45	DFHTQGDS	TDQUEUE (globals) id
STITSQ	48	DFHTSGDS	TSQUEUE statistics id
STICONSR	52	DFHA14DS	ISC/IRC system entry (resid) id
STICONSS	54	DFHA21DS	ISC connection - system security
STIUSG	61	DFHUSGDS	User domain stats id
STIDS	62	DFHDSGDS	Dispatcher stats id
STITM	63	DFHA16DS	Table manager statistics id
STIDST	64	DFHDSTDS	Dispatcher TCB (global)id
STIDSR	65	DFHDSRDS	Dispatcher TCB (resid)id
STIST	66	DFHSTGDS	Statistics statistics id
STIFCR	67	DFHA17DS	File Control (resid) id
STIMQG	74	DFHMQGDS	MQ connection stats (global) id
STICONMR	76	DFHA20DS	ISC/IRC mode entry (resid) id
STIM	81	DFHMNGDS	Monitoring stats (global) id
STIMNR	84	DFHMNTDS	Monitoring stats (Resid) id
STITDR	85	DFHTDRDS	Transaction dump (resid) id
STITDG	87	DFHTDGDS	Transaction dump (global) id
STISDR	88	DFHSDRDS	System dump (resid) id
STISDG	90	DFHSDGDS	System dump (global) id
STILGG	92	DFHLGGDS	Logstream stats (global) id
STILGR	93	DFHLGRDS	Logger stats (resid) id
STILGS	94	DFHLGSDS	Logstream stats (resid) id
STINQG	97	DFHNQGDS	Enqueue mgr stats (global) id
STIRMG	99	DFHRMGDS	Recovery mgr stats (global) id
STIRLR	100	DFHRLRDS	BUNDLEs (resource) id
STIWBG	101	DFHWBGDS	URIMAPs (global) id
STID2G	102	DFHD2GDS	DB2 connection stats (global) id
STID2R	103	DFHD2RDS	DB2 entry stats (resource) id
STIWBR	104	DFHWBRDS	URIMAPs (resource) id
STIPIR	105	DFHPIRDS	PIPELINE (resource) id
STIPIW	106	DFHPIWDS	WEBSERVICE (resource) id
STISOG	107	DFHSOGDS	TCP/IP (global) id
STISOR	108	DFHSORDS	TCPIP services (resource) id
STIISR	109	DFHISRDS	IPCONN (resource) id
STIW2R	110	DFHW2RDS	ATOMSERVICE (resource) id
STIDHD	112	DFHDHDDS	DOCTEMPLATE (resource) id
STIMLR	113	DFHMLRDS	XMLTRANSFORM (resource) id

STID Symbolic name	STID Value	Copybook	Type of record
STISJS	116	DFHSJSDS	JVMSERVER stats (resource) id
STIPGR	119	DFHPGRDS	JVMPROGRAM stats (resource) id
STIPGD	120	DFHPGDDS	PROGRAMDEF stats (resource) id
STIECG	140	DFHECGDS	EVENTBINDINGs (global) id
STIECR	141	DFHECRDS	EVENTBINDINGs (resource) id
STIEPG	142	DFHEPGDS	EVENTPROCESS (global) id
STIECC	143	DFHECCDS	CAPTURESPECs (resource) id
STIEPR	144	DFHEPRDS	EPADAPTERs (resource) id
STIPGP	146	DFHPGPDS	JVM programs - private
STIPGE	147	DFHPGEDS	Program definitions - private

Figure 81. Statistics data record copybooks related to STID name and value

The TS data sharing statistics use no symbolic names, but relate to the STID values as follows:

```
STID      STID
Symbolic  Value  Copybook     Type of record
name

 -         121   DFHXQS1D      TS server list structure stats id
 -         122   DFHXQS2D      TS buffer stats id
 -         123   DFHXQS3D      TS storage stats id
```

Figure 82. TS data sharing statistics related to STID

The coupling facility data table server statistics use no symbolic names, but relate to the STID values as follows:

```
STID      STID
Symbolic  Value  Copybook     Type of record
name

 -         126   DFHCFS6D      CFDT server list stats
 -         127   DFHCFS7D      CFDT buffer stats id
 -         128   DFHCFS8D      CFDT request stats id
 -         129   DFHCFS9D      CFDT storage stats id
```

Figure 83. Coupling facility data table server statistics related to STID

The named sequence number server statistics use no symbolic names, but relate to the STID values as follows:

```
STID      STID
Symbolic  Value  Copybook     Type of record
name

 -         124   DFHNCS4D      NC server list structure stats id
 -         125   DFHNCS5D      NC server storage stats id
```

Figure 84. Named sequence server statistics related to STID

Chapter 27. Using an XSTOUT global user exit program to filter statistics records

About this task

There is one global user exit point (XSTOUT) in the CICS statistics domain. The exit is started before the contents of a statistics data buffer are written to SMF. At this exit, the following information is available:

- The address of the statistics buffer
- The length of the statistics buffer
- The address of the statistics type.

This applies to all five types of statistics: interval, end-of-day, requested, requested reset, and unsolicited statistics.

If you write a global user exit program to be started at this exit, you can examine this information and tell CICS either to write the contents of the buffer to SMF or to suppress its output.

For more information about global user exits in general, and about the statistics exit in particular, refer to Chapter 1, "Global user exit programs," on page 3.

Chapter 28. Processing the output from CICS statistics

You have several options for processing statistics output.

You can use:

The CICS-supplied DFHSTUP program
For information about how to run DFHSTUP, refer to Statistics utility program (DFHSTUP) in Reference -> Utilities. For information about how to interpret the report produced by DFHSTUP, see CICS statistics in DSECTS and DFHSTUP report in Monitoring.

Tivoli® Decision Support for z/OS
enables you to store CICS statistics (and other data) into a DB2 data set, and to present the data in a variety of forms. For information about Tivoli Decision Support for z/OS, see Tivoli Decision Support for z/OS in Improving performance.

Your own program
to report and analyze the data in the statistics records. The format of CICS statistics records is described in Chapter 26, “CICS statistics record format,” on page 653.

Part 5. Structure and content of CICS TS format journal records

SMF records

The following description does not apply to journal records written to an SMF data set. These are described Part 7, “Format of journal records written to SMF,” on page 697.

System logs are always presented in CICS TS format.

Each general log comprises a stream of contiguous blocks of journaled data. Each block comprises a block header followed by a variable number of CICS journal records. Each CICS journal record comprises a record header followed by caller data.

Figure 85 on page 664 gives a graphical overview of a general log, showing the format of a complete block, and the format of a complete journal record.

The format of the caller data depends on the CICS component that is issuing the journal record, and also on the function being journaled at the time. Thus, for example, the format of caller data in journal records issued by file control differs from that of caller data in journal records issued by FEPI.

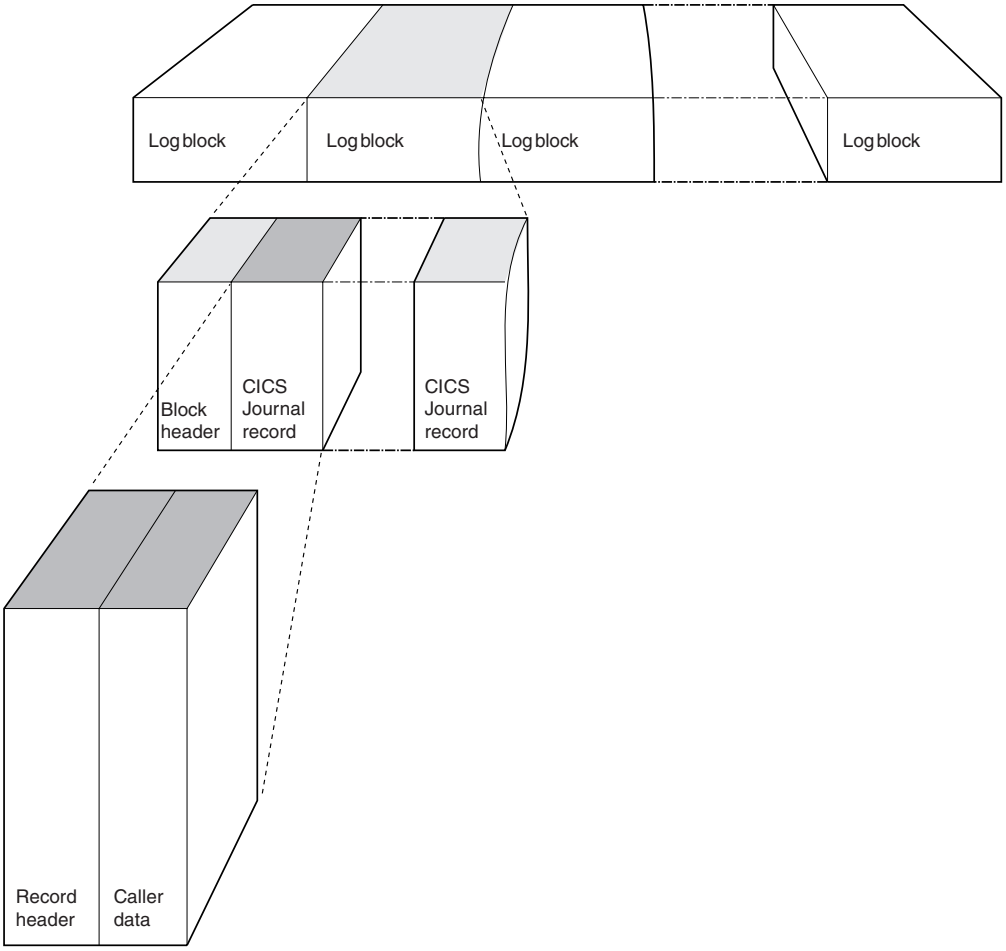
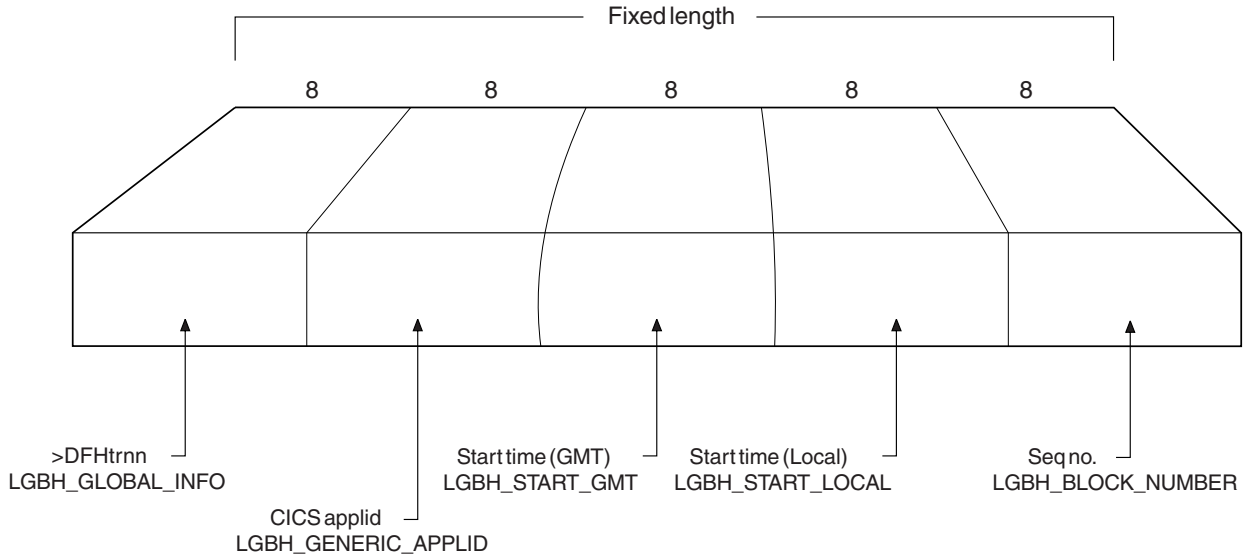


Figure 85. Layout of a general log

Chapter 29. Format of general log block header

The log block header contains information of a general system-wide nature such as the CICS applid writing the journal block. Figure 86 shows the format of the log block header.



Where *t = Log type*
r = reserved
nn = block version number

Figure 86. Format of a general log block header

LGBH_GLOBAL_INFO
8-bytes containing '>DFHtrnn', where:

```
t  = log type
r  = reserved
nn = block version number
```

LGBH_GENERIC_APPLID
8-byte CICS applid.

LGBH_START_GMT
8-byte start time (GMT).

LGBH_START_LOCAL
8-byte start time (local).

LGBH_BLOCK_NUMBER
8-byte sequence number.

Chapter 30. Format of general log journal record

The journal record comprises a record header followed by caller data. The record header contains information that describes some of the attributes of the record, such as the time it was written. Figure 87 shows the format of the record header.

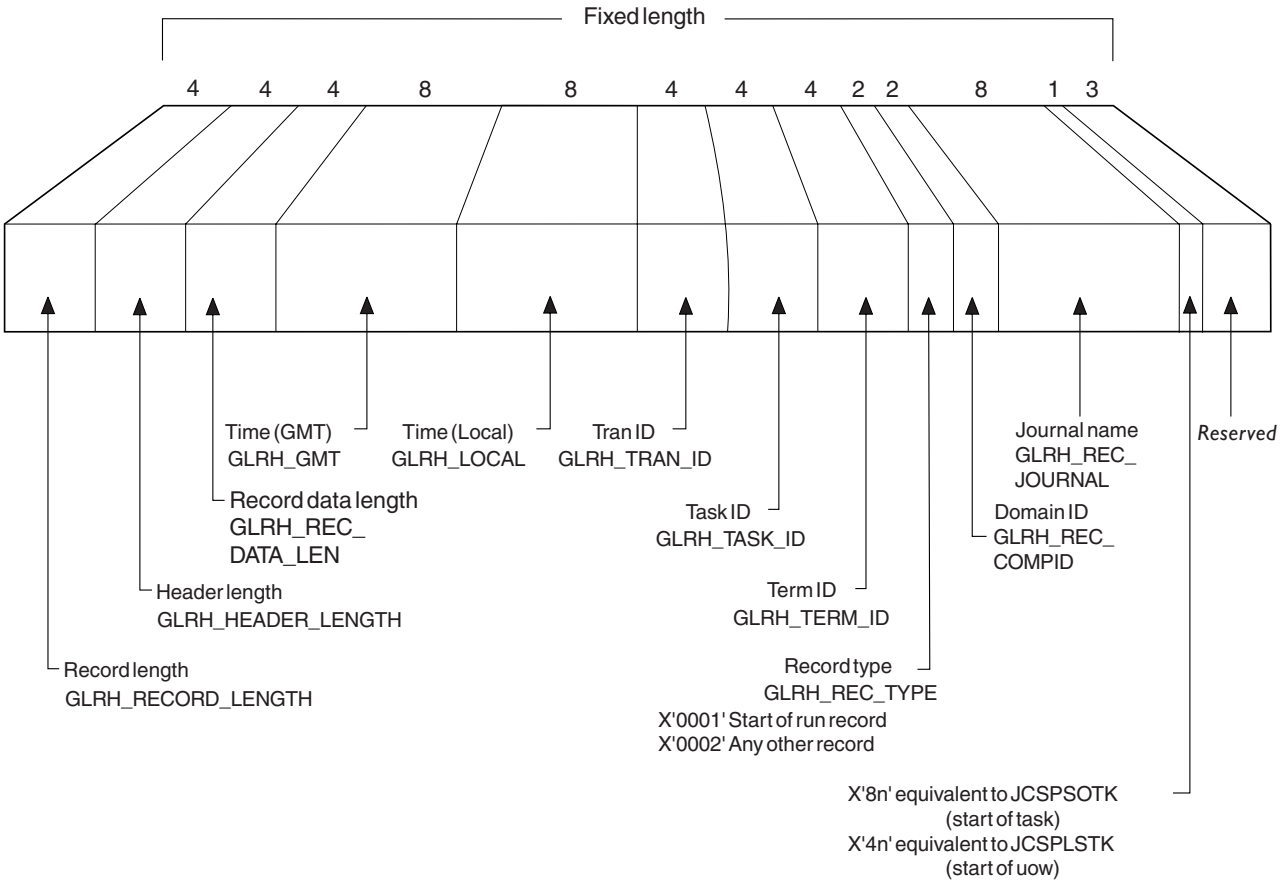


Figure 87. Format of a general log record header

GLRH_RECORD_LENGTH
4-byte record length.
GLRH_HEADER_LENGTH
4-byte header length.
GLRH_REC_DATA_LEN
4-byte record data length.
GLRH_GMT
8-byte time (GMT).
GLRH_LOCAL
8-byte time (local).
GLRH_TRAN_ID
4-byte transaction identifier.
GLRH_TASK_ID
4-byte task identifier.
GLRH_TERM_ID
4-byte terminal identifier.

GLRH_REC_TYPE
2-byte record type. Either:

```
X'0001'  Start of run record
X'0002'  Any other record
```

GLRH_REC_COMPID
2-byte domain identifier.

GLRH_REC_JOURNAL
8-byte journal name.

Start of task indicator
1-byte which may contain:

```
X'8n'   Equivalent to JCSPSOTK (start of task)
X'4n'   Equivalent to JCSPLSTK (start of UOW)
```

Reserved
3-byte reserved field.

The caller data differs depending on the CICS component issuing the record, and on the function being journaled.

Chapter 31. Start-of-run record

When CICS connects to a general log, it writes a start-of-run record to it as the first record for this run of CICS. This record comprises a record header (with the same format as that for any general log journal record) followed by a start-of-run body

. The format of the start-of-run record is shown in Figure 88.

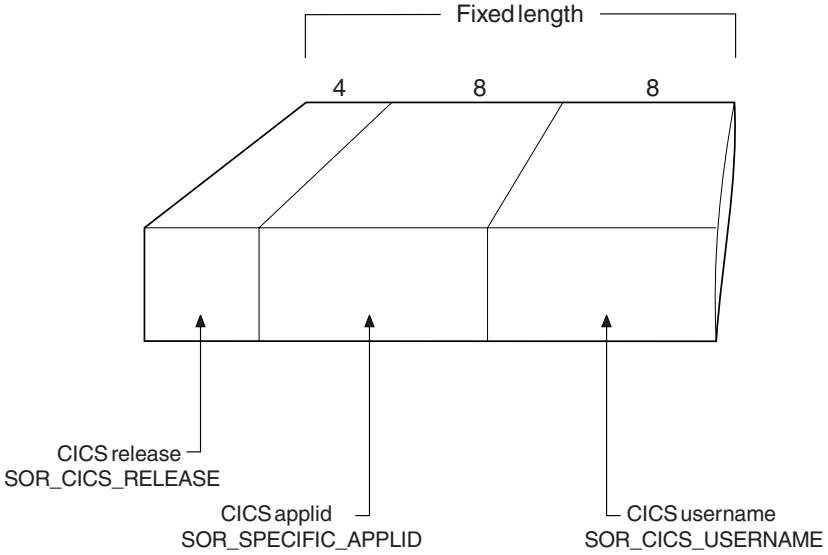


Figure 88. Format of the start-of-run record

SOR_CICS_RELEASE
4-byte CICS release.
SOR_SPECIFIC_APPLID
8-byte CICS applid.
SOR_CICS_USERNAME
8-byte CICS username.

For a start-of-run record, CICS puts the domain identifier "LG" (for "logger") in the GLRH_REC_COMPID field of the record header.

Chapter 32. Format of caller data

Caller data follows the record header. The format of the caller data part of a general log journal record differs according to the CICS component writing the record, and the function being journaled.

Journal records can be written by any of the following CICS components: the logger, journal control (in the case of a request issued by a user), file control, the front end programming interface (FEPI), and terminal control. The field GLRH_REC_COMPID in the record header tells you which component has written the record: LG, UJ, FC, SZ, or TC respectively.

File control adds information to the start of the actual journaled data, and this is described in "Caller data written by file control" on page 672. The other components (journal control, FEPI, and terminal control) do not add any further information to the journaled data.

If the record has been written by the CICS API, the caller data section starts with an API user header, the format of which is shown in Figure 89.

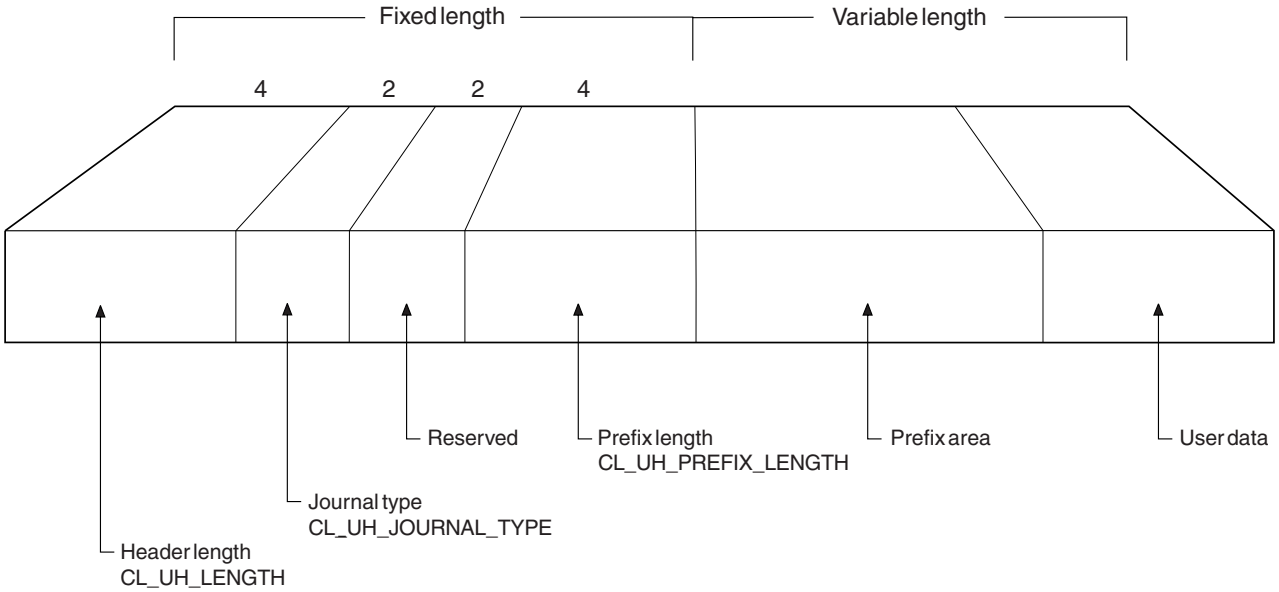


Figure 89. Format of the API user header

CL_UH_LENGTH
4-byte header length.
CL_UH_JOURNAL_TYPE
2-byte journal type.
Reserved
2-byte reserved field.
CL_UH_PREFIX_LENGTH
4-byte prefix length.
Prefix Variable-length prefix area.
User data
Variable-length user data.

Caller data written by file control

The file log and journal block (FLJB) describes the caller data that file control writes as part of its journal records. The copybook DFHFCLGD defines the FLJB DSECT.

There are two sections in the FLJB: the first section contains data that is applicable to all journal records written by file control; the second section contains information specific to the record type. Both sections are of fixed length.

Some records have a third and fourth section which are of variable length.

Table 36 outlines the sections in a journal record written by file control.

Table 36. FLJB sections in journal records issued by file control

Record type	First section	Second section	Third section	Fourth section
Read-only Read-update Write-update Write-add Write-add complete	FLJB_GENERAL_ DATA	FLJB_COMMON_DATA	FLJB_CD_KEY	FLJB_CD_DATA
Write delete	FLJB_GENERAL_ DATA	FLJB_WRITE_DELETE _DATA	FLJB_WDD_BASE _KEY	FLJB_WDD_PATH _KEY
File close	FLJB_GENERAL_ DATA	FLJB_FILE_CLOSE _DATA	None	None
Tie-up	FLJB_GENERAL_ DATA	FLJB_TIE_UP_RECORD _DATA	None	None

A description of each of the structures, and the sections within them, now follows:

Read-only, read-update, write-update, write-add, write-add complete record types

There are four sections in the journal records written for read-only, read-update, write-update, write-add, and write-add complete record types:

- The FLJB_GENERAL_DATA section,
- The FLJB_COMMON_DATA section, and
- The caller data image sections which consist of the FLJB_CD_KEY (the length of which is given in FLJB_COMMON_DATA) and the FLJB_CD_DATA section. The FLJB_CD_DATA section (the length of which is given in FLJB_COMMON_DATA) contains the image of the caller data.

The format of such a record written for these record types is shown in Figure 90 on page 673.

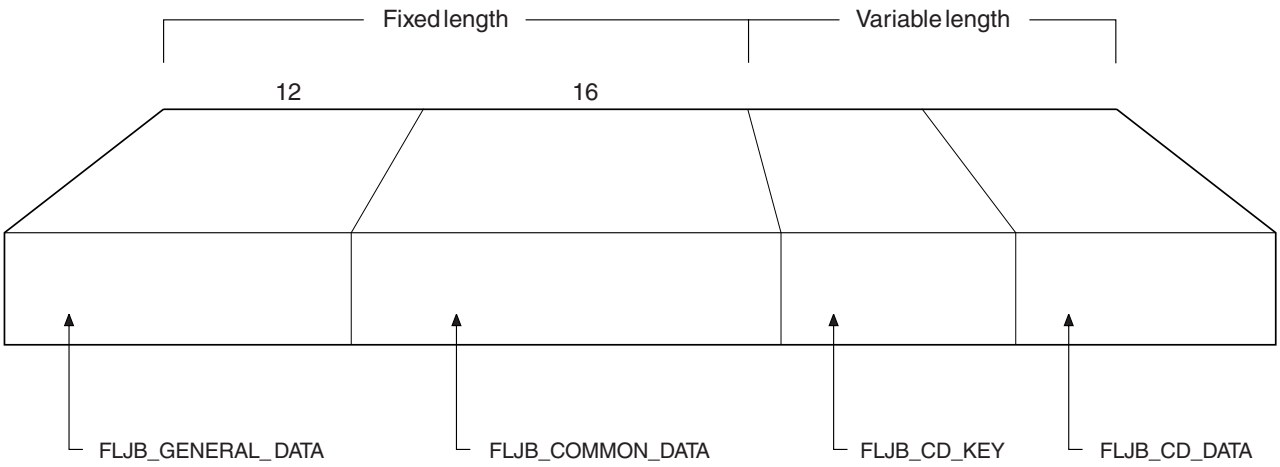


Figure 90. Layout of record written for read-only, read-update, write-update, write-add, and write-add-complete record types

FLJB_GENERAL_DATA
12-byte general data.
FLJB_COMMON_DATA
16-byte common data.
FLJB_CD_KEY
Variable-length caller data key.
FLJB_CD_DATA
Variable-length caller data.

The format of the FLJB_GENERAL_DATA section is shown in Figure 91.

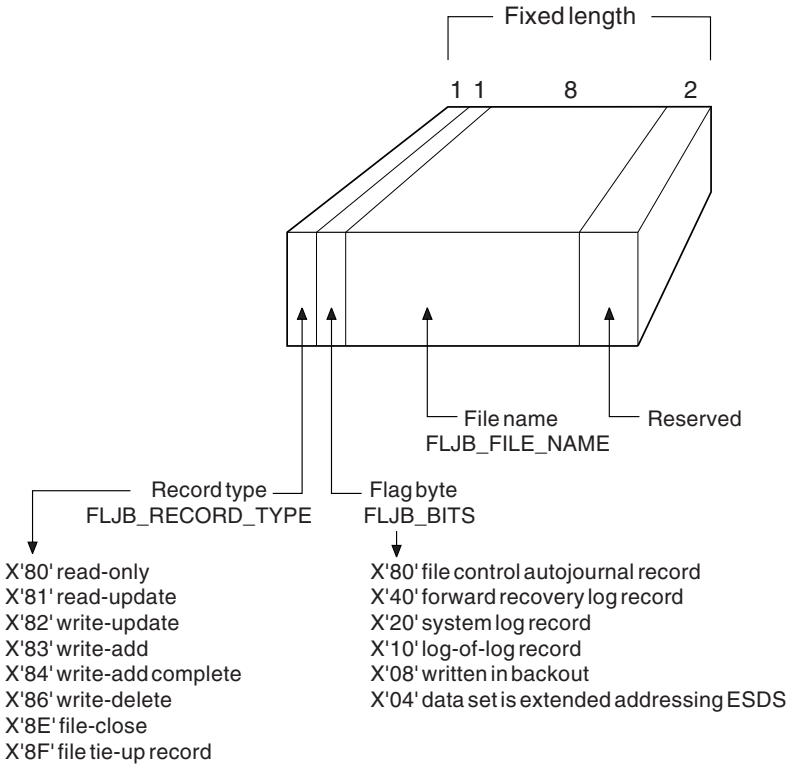


Figure 91. Format of FLJB_GENERAL_DATA section

FLJB_RECORD_TYPE
1-byte record type:

```
X'80'    Readonly
X'81'    Read update
X'82'    Write update
X'83'    Write add
X'84'    Write add complete
X'86'    Write delete
X'8E'    File close
X'8F'    File tie-up record
```

FLJB_BITS
1-byte flag field:

```
X'80'    File control autojournal record
X'40'    Forward recovery log record
X'20'    System log record
X'10'    Log-of-log record
X'08'    Written in backout
X'04'    Data set is extended addressing ESDS
```

FLJB_FILE_NAME
8-byte file name.

Reserved
2-byte reserved field.

The format of the FLJB_COMMON_DATA section is shown in Figure 92.

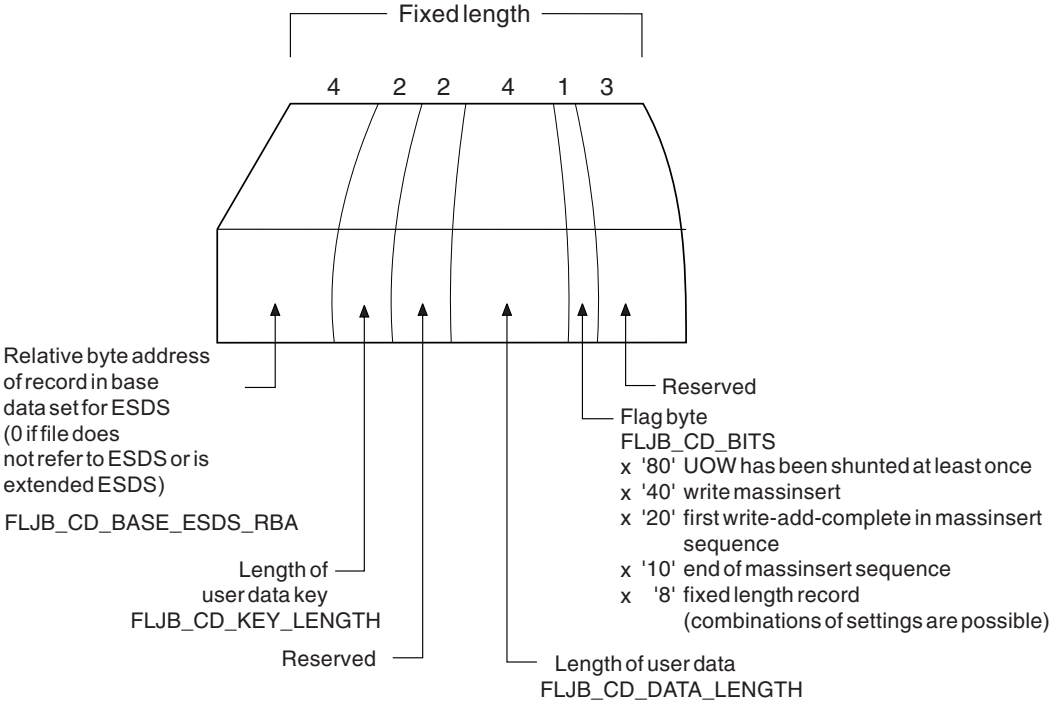


Figure 92. Format of FLJB_COMMON_DATA section

FLJB_CD_BASE_ESDS_RBA
4-byte relative byte address of record in base data set for ESDS (0 if file does not refer to ESDS, or if is an extended addressing ESDS).

FLJB_CD_KEY_LENGTH
2-byte length of user data key. (The key length is 4 for an RRDS, a VRRDS, or a standard ESDS; 8 for an extended addressing ESDS.)

Reserved
2-byte reserved field.

FLJB_CD_DATA_LENGTH
4-byte length of user data.
FLJB_CD_BITS
1-byte flag field:

```
X'80'    UOW has been shunted at least once
X'40'    Write massinsert
X'20'    First write-add-complete in massinsert sequence
X'10'    End of massinsert sequence
X'8'     Fixed length record
```

Combinations of settings are possible.
Reserved
3-byte reserved field.

Write-delete record types

There are four sections in the journal records written for write-delete record types:

- The FLJB_GENERAL_DATA section,
- The FLJB_WRITE_DELETE_DATA section, and
- The two caller data image sections, which consist of:
  - FLJB_WDD_BASE_KEY (the length of which is given by FLJB_WDD_BASE_KEY_LENGTH in FLJB_WRITE_DELETE_DATA)
  - FLJB_WDD_PATH_KEY (the length of which is given by FLJB_WDD_PATH_KEY_LENGTH in FLJB_WRITE_DELETE_DATA).

  These sections contain the image of the caller data as the base key, and, if the data set is a path, the path.

The format of such a record written for write-delete record types is shown in Figure 93.

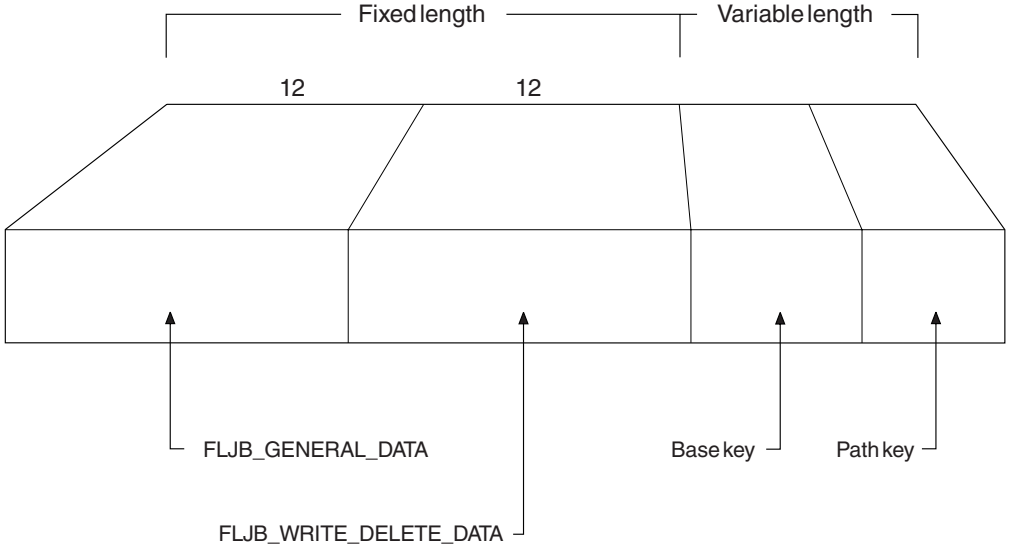


Figure 93. Layout of record written for write-delete record types

FLJB_GENERAL_DATA
12-byte general data section.
FLJB_WRITE_DELETE_DATA
12-byte write-delete data section.

FLJB_WDD_BASE_KEY
Variable-length base key.
FLJB_WDD_PATH_KEY
Variable-length path key.

See Figure 91 on page 673 for the format of the FLJB_GENERAL_DATA section.

The format of the FLJB_WRITE_DELETE_DATA section is shown in Figure 94.

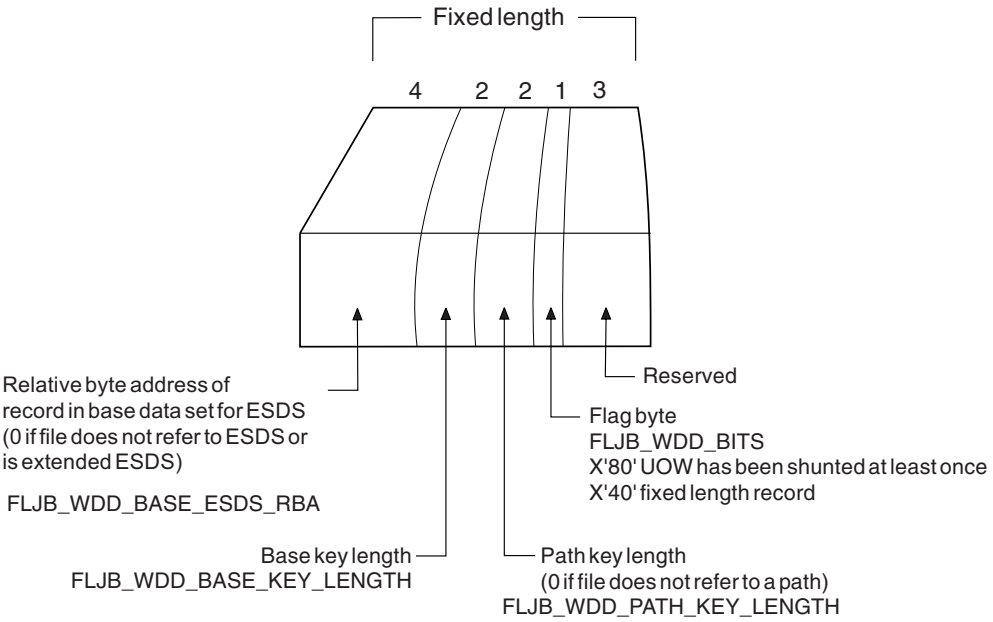


Figure 94. Format of the FLJB_WRITE_DELETE_DATA section

FLJB_WDD_BASE_ESDS_RBA
4-byte relative byte address of record in base data set for ESDS (0 if file does not refer to ESDS, or if it is an extended addressing ESDS).
FLJB_WDD_BASE_KEY_LENGTH
2-byte base key length. (The key length is 4 for an RRDS, a VRRDS, or a standard ESDS; 8 for an extended addressing ESDS.)
FLJB_WDD_PATH_KEY_LENGTH
2-byte path key length (0 if file does not refer to a path).
FLJB_WDD_BITS
1-byte flag field:

```
X'80'    UOW has been shunted at least once
X'40'    Fixed-length record
```

Reserved
3-byte reserved field.

File-close record types

There are two sections in the journal records written for file-close record types:

- The FLJB_GENERAL_DATA section
- The FLJB_CLOSE_DATA section.

The format of such a record written for file-close record types is shown in Figure 95.

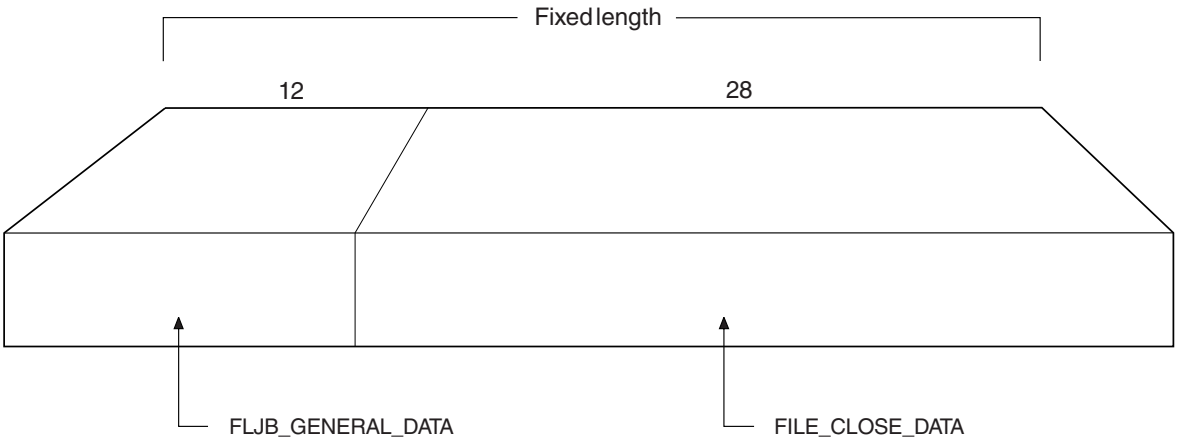


Figure 95. Layout of record written for file-close record types

FLJB_GENERAL_DATA
12-byte general data section.
FLJB_CLOSE_DATA
28-byte close data section.

See Figure 91 on page 673 for the format of the FLJB_GENERAL_DATA section.

The format of the FLJB_CLOSE_DATA section is shown in Figure 96.

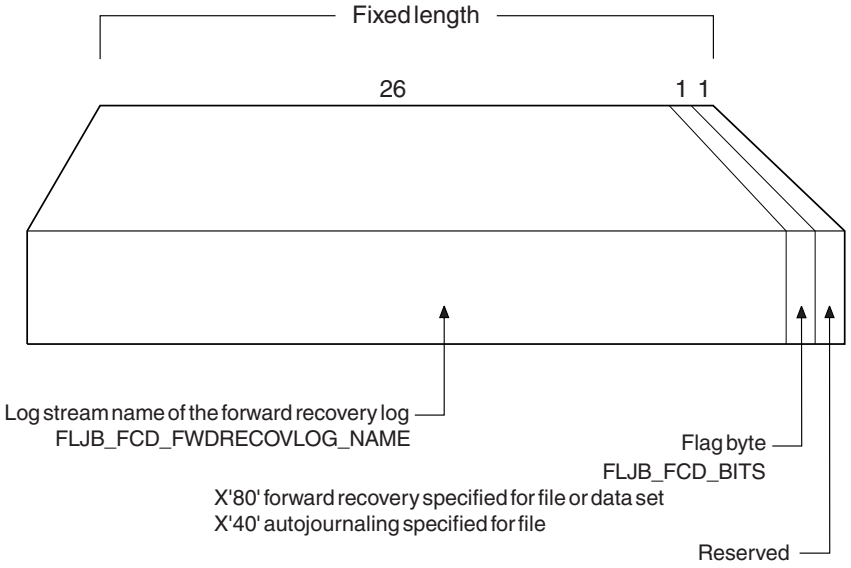


Figure 96. Format of the FILE_CLOSE_DATA section

FLJB_FCD_FWDRECOVLOG_NAME
26-byte log stream name of the forward recovery log.
FLJB_FCD_BITS
1-byte flag field:

```
X'80'    Forward recovery specified for file or data set
X'40'    Autojournaling specified for file
```

Reserved
1-byte reserved field.

Tie-up record types

There are two sections in the journal records written for tie-up record types:

- The FLJB_GENERAL_DATA section
- The TIE_UP_RECORD_DATA section.

The format of such a record written for tie-up record types is shown in Figure 97.

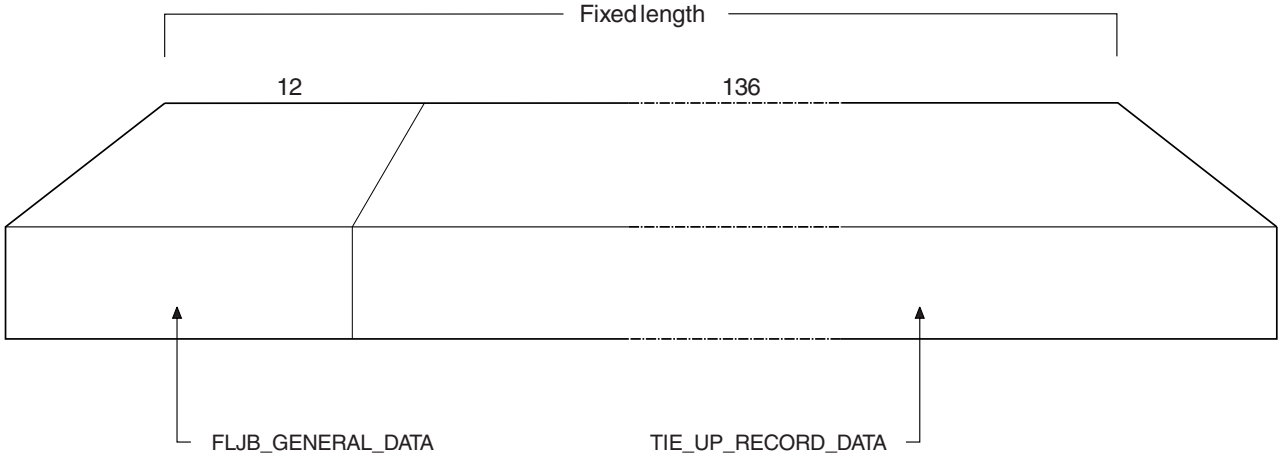


Figure 97. Layout of record written for tie-up record types

FLJB_GENERAL_DATA
12-byte general data.
TIE_UP_RECORD_DATA
136-byte tie-up record data.

See Figure 91 on page 673 for the format of the FLJB_GENERAL_DATA section.

The format of the TIE_UP_RECORD_DATA section is shown in Figure 98 on page 679.

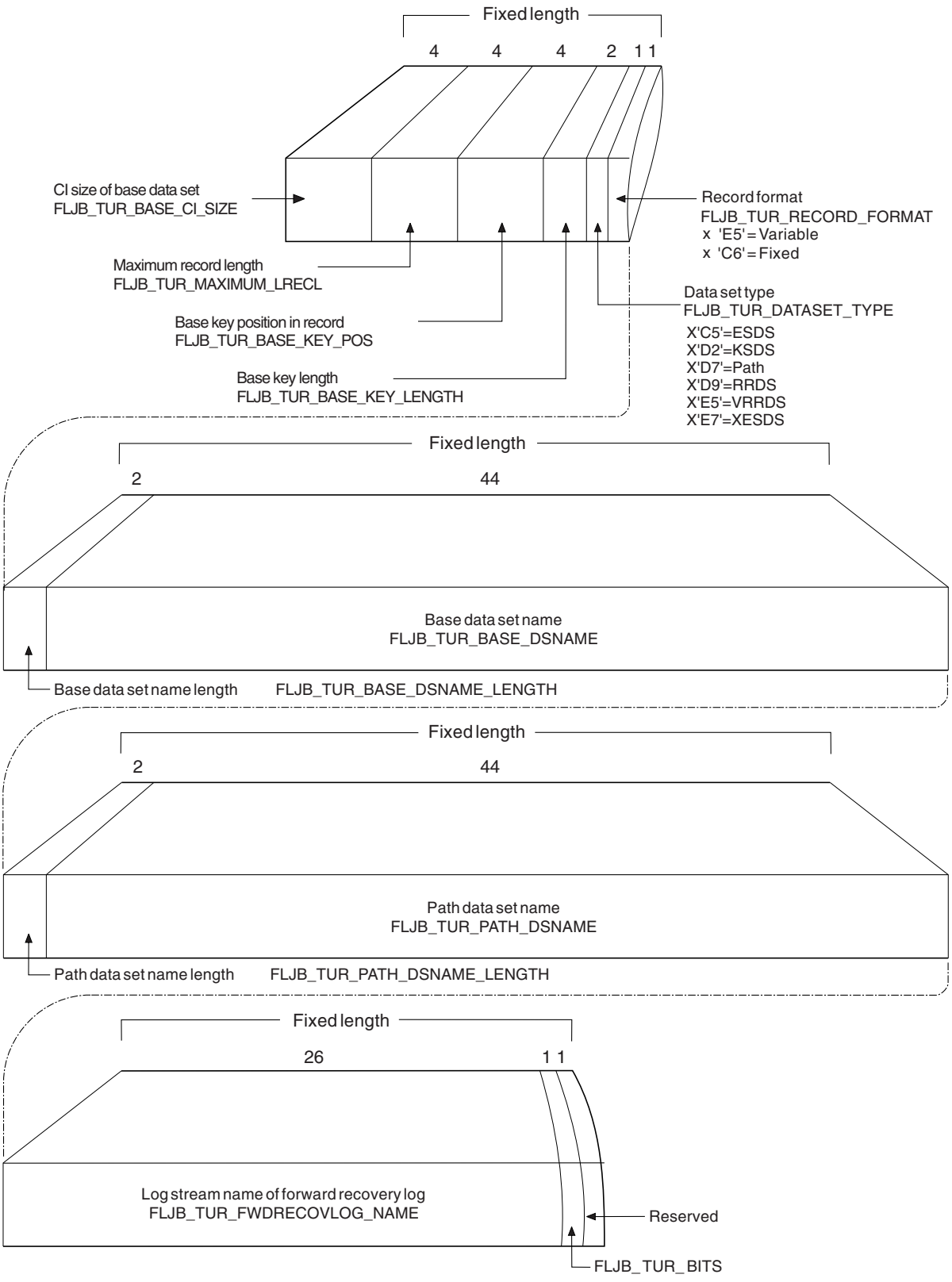


Figure 98. Format of TIE_UP_RECORD_DATA section

FLJB_TUR_BASE_CI_SIZE
4-byte CI size of base data set.

FLJB_TUR_MAXIMUM_LRECL
4-byte maximum record length.
FLJB_TUR_BASE_KEY_POSITION
4-byte base key position in record.
FLJB_TUR_BASE_KEY_LENGTH
2-byte base key length.
FLJB_TUR_DATASET_TYPE
1-byte data set type:

```
X'C5'     Standard ESDS
X'D2'     KSDS
X'D7'     Path
X'D9'     RRDS
X'E5'     VRRDS
X'E7'     Extended ESDS
```

FLJB_TUR_RECORD_FORMAT
1-byte record format:

```
X'E5'     Variable
X'C6'     Fixed
```

FLJB_TUR_BASE_DSNAME_LENGTH
2-byte length of base data set name.
FLJB_TUR_BASE_DSNAME
44-byte base data set name.
FLJB_TUR_PATH_DSNAME_LENGTH
2-byte length of path data set name.
FLJB_TUR_PATH_DSNAME
44-byte path data set name.
FLJB_TUR_FWDRECOVLOG_NAME
26-byte log stream name of forward recovery log.
FLJB_TUR_BITS
1-byte flag field.
Reserved
1-byte reserved field.

Note: The format of caller data in journal records written by file control in RLS mode is identical to that in journal records written by file control in non-RLS mode except for FLJB_TUR_BITS where a value of X'80' indicates RLS-access.

Terminal control prefix data

CICS terminal control (TC) writes journal records to track the messages it issues. Each TC journal record contains a prefix area, which lies in the position of the prefix area in the API user header.

For LU6.1-related records only, the prefix area contains the VTAM® physical sequence numbers at syncpoint time; for all other TC journal records, it contains binary zeros. The format of the TC prefix area is shown in Figure 99 on page 681.

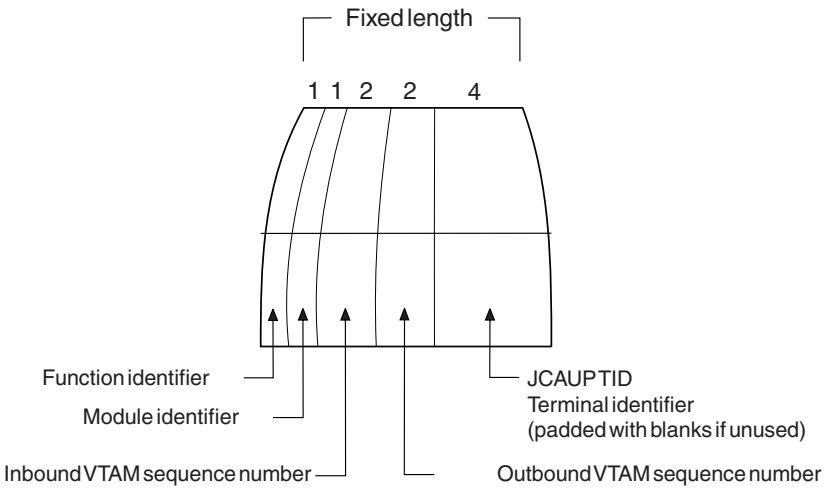


Figure 99. Format of the terminal control prefix area

Function ID
1-byte function identifier.
Module ID
1-byte module identifier.
Inbound VTAM SN
2-byte inbound VTAM sequence number.
Outbound VTAM SN
2-byte outbound VTAM sequence number.
JCAUP TID
4-byte terminal identifier (padded with blanks if unused).

FEPI prefix data

Each FEPI journal record contains a prefix area that allows you to identify the FEPI conversation for which the data was journaled.

This prefix area lies in the position of the prefix area in the API user header. Its format is shown in Figure 100 on page 682.

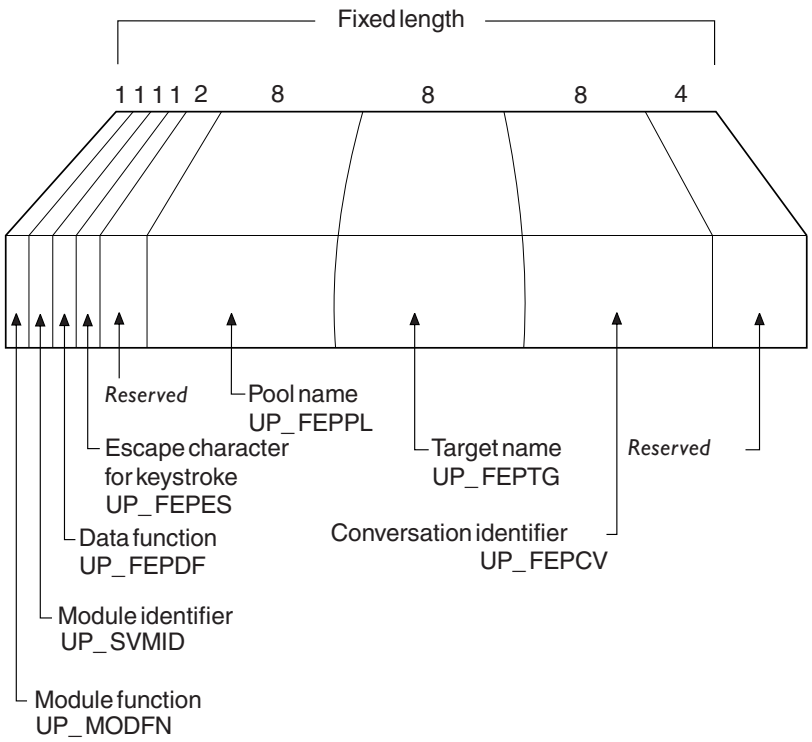


Figure 100. Format of the FEPI prefix area

UP_MODFN
1-byte module function

UP_SVMID
1-byte module identifier

UP_FEPDF
1-byte data function

UP_FEPES
1-byte escape character for keystroke

Reserved
2-byte reserved field

UP_FEPPL
8-byte pool name

UP_FEPTG
8-byte target name

UP_FEPCV
8-byte conversation identifier

Reserved
4-byte reserved field

Part 6. Structure and content of COMPAT41-format journal records

CICS allows you to format journal records so that they are presented in the format used at CICS/ESA 4.1. Use the COMPAT41 option on the SUBSYS=(LOGR...) step of your JCL.

SMF records

The following description does not apply to journal records written to an SMF data set. These are described in Part 7, "Format of journal records written to SMF," on page 697.

Within the data, certain fields are not presented. They appear as X'00' in the formatted output. These fields are:

Table 37. Fields formatted as X'00'

JCLRJFID	JCLRTBAL	JCSPEMER
JCLRVCD	JCRBB	JCSPMIDT
JCLRVSN	JCSPFS	JCSPRRIF
JCLRLBW	JCSPDSP	

Each general log comprises a stream of contiguous blocks of journaled data. Each block comprises a journal control label header followed by a variable number of CICS journal records. Each CICS journal record comprises a system header, system prefix, user prefix, and journaled data.

A graphical overview of the format of a general log, showing the format of a complete block, is shown in Figure 101.

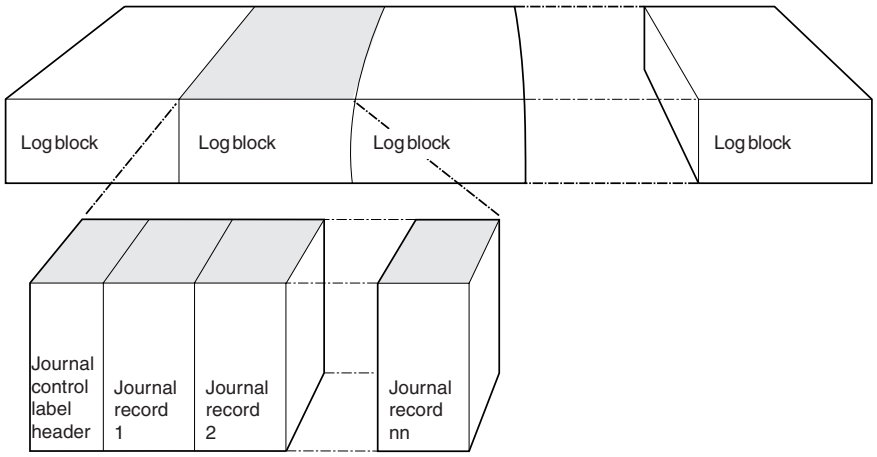


Figure 101. Format of general log formatted using the COMPAT41 option

Chapter 33. Format of COMPAT41 journal control label header

Each log block starts with a journal control label header.

There is one journal control label header per log block. It is 42 bytes long, and comprises a length field, label header, and label prefix. The format of the journal control label header is shown in Figure 102.

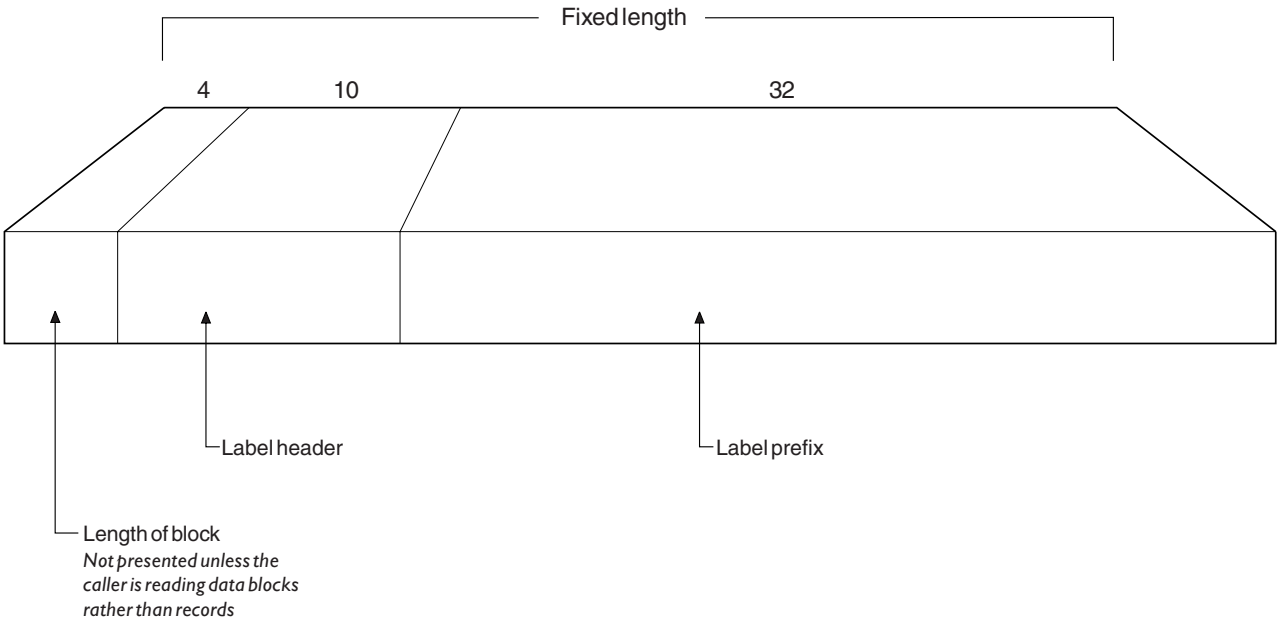


Figure 102. Format of journal control label header

The label header part of the journal control label header is 10 bytes long, and its format is shown in Figure 103 on page 686.

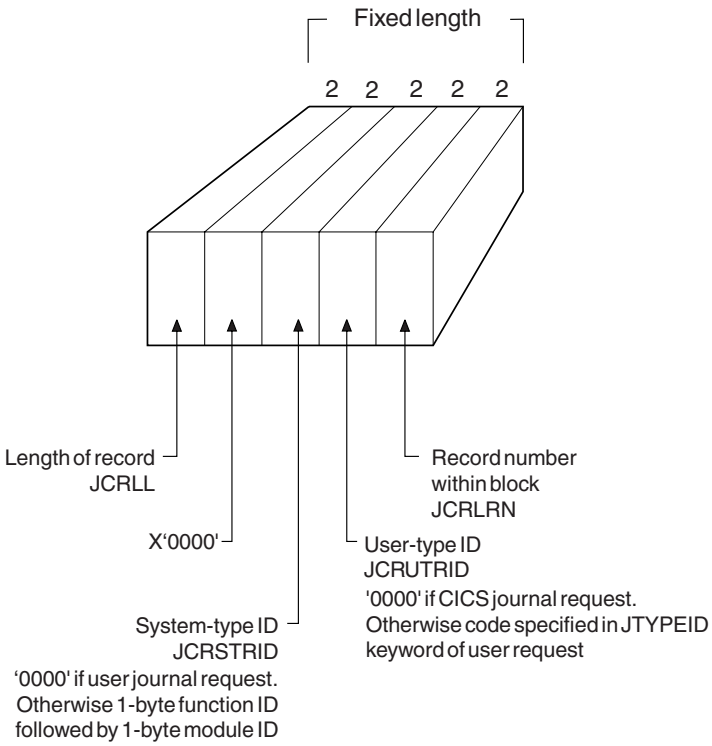


Figure 103. Format of label header part of journal control label header

JCRLL
2-byte length of record.

X'0000'
2-bytes containing X'0000'.

JCRSTRID
2-byte system-type identifier. For a user journal request, this is X'0000'. Otherwise, it consists of a 1-byte function ID followed by a 1-byte module ID.

JCRUTRID
2-byte user-type identifier. For a CICS journal request, this is X'0000'. Otherwise, it contains the code specified by the JTYPEID keyword of the user request.

JCRLRN
2-byte record number within the block.

The label prefix part of the journal control label header is 32 bytes long, and its format is shown in Figure 104 on page 687.

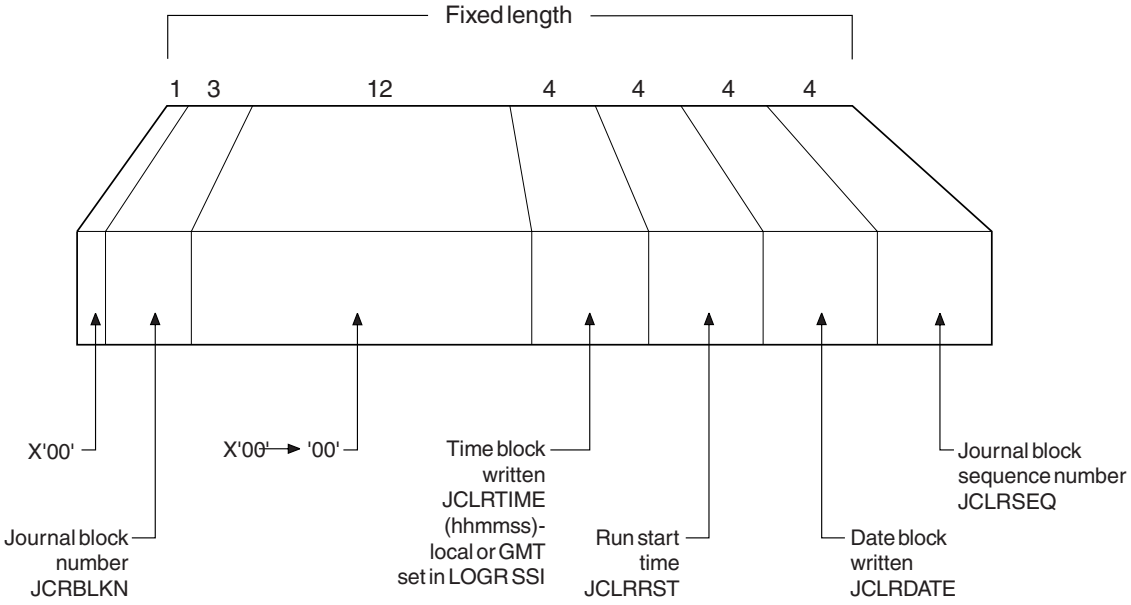


Figure 104. Format of label prefix part of journal control label header

X'00' 1-byte containing X'00'.

JCRBLKN
3-byte journal block number.

X'00's 12-bytes each containing X'00'.

JCLRTIME
4-bytes containing the time the block was written, in hhmmss format. (Local or GMT set in LOGR SSI.)

JCLRRST
4-bytes containing the run start time.

JCLRDATE
4-bytes containing the date the block was written.

JCLRSEQ
4-byte journal block sequence number.

Chapter 34. Format of journal record

Each CICS journal record comprises a system header, system prefix, user prefix, and journaled data.

The format of a journal record is shown in Figure 105.

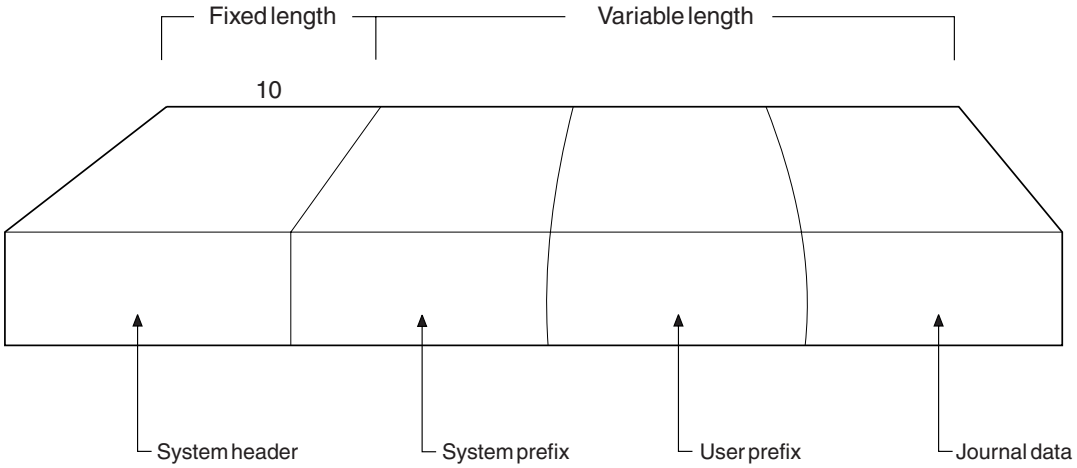


Figure 105. Format of COMPAT41 journal record

The system header is 10 bytes long. Its format is shown in Figure 106.

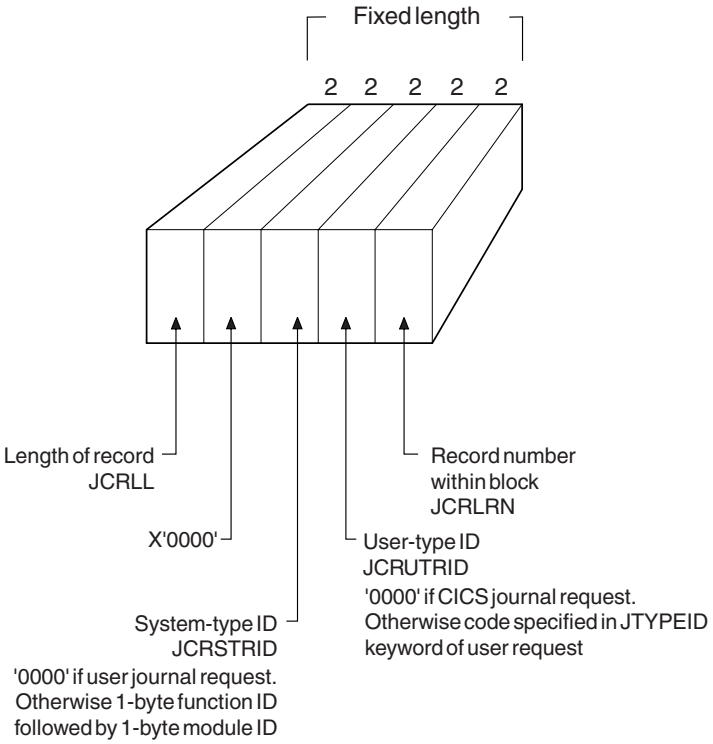


Figure 106. Format of the system header

JCRLL
2-byte length of record.

X'0000'
2-bytes containing X'0000'.

JCRSTRID
2-byte system-type identifier. For a user journal request, this is X'0000'. Otherwise, it consists of a 1-byte function ID followed by a 1-byte module ID.

JCRUTRID
2-byte user-type identifier. For a CICS journal request, this is X'0000'. Otherwise, it contains the code specified by the JTYPEID keyword of the user request.

JCRLRN
2-byte record number within the block.

The field JCRSTRID (the system-type ID) and the field JCRUTRID (the user-type ID) in the system header allow you to distinguish those journal records output by CICS (by such components as terminal control), from those issued by direct user requests.

For CICS journal requests, JCRUTRID contains binary zeros, and JCRSTRID contains a 1-byte function code followed by a 1-byte module code. The function code tells you which function was being journaled, and the module code shows which module caused the record to be written. Valid settings of these codes are contained in the member DFHFMIDS of the CICS assembler-language macro library. Figure 109 on page 693 shows the valid function identifiers of those CICS components that issue journal requests. Figure 110 on page 694 shows the valid module identifiers.

For user journal requests, JCRSTRID always contains binary zeros, and JCRUTRID contains the 2-byte hexadecimal code specified by the JTYPEID keyword of the WRITE JOURNALNAME request in the application program.

The system prefix is 20 bytes long. Its format is shown in Figure 107 on page 691.

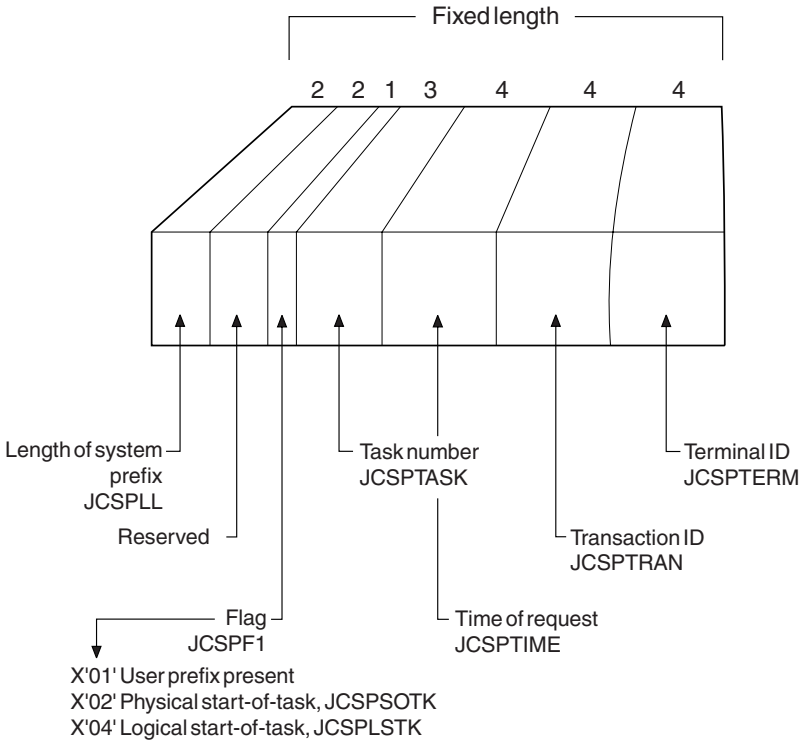


Figure 107. Format of the system prefix

JCSPLL
2-byte length of the system prefix.

Reserved
2-byte reserved field.

JCSPF1
1-byte flag field:

```
X'01'    User prefix present
X'02'    Physical start-of-task, JCSPSOTK
X'04'    Logical start-of-task, JCSPLSTK
```

JCSPTASK
3-byte task number.

JCSPTIME
4-byte time of request.

JCSPTRAN
4-byte transaction identifier.

JCSPTERM
4-byte terminal identifier.

For some CICS journal requests, additional data is included in the system prefix to identify more specifically the originator of the request. This extra data follows the common fields of the system prefix, and is usually variable in length; hence the need for the length field JCSPLL at the start of the system prefix. All the following have their own prefix layout, and these are described, for the purposes of diagnosis and recovery, in the *CICS Data Areas* manual.

The user prefix is a variable length area. It is present if this record has been written by a user request, an EXEC CICS WRITE JOURNALNAME command. The information contained in the record is set by the user within the terms of the command via the JTYPEID, PREFIX, and PFXLENG parameters. Its format is

shown in Figure 108.

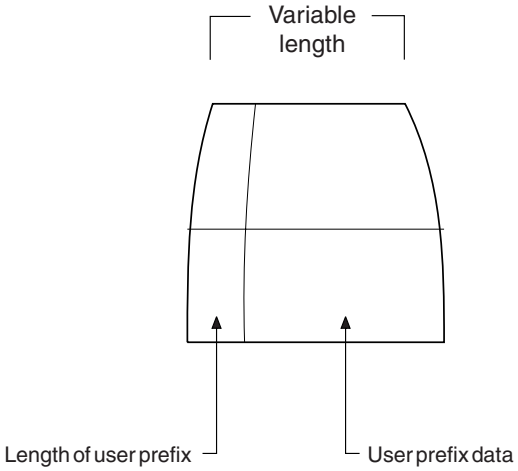


Figure 108. Format of the user prefix

Field JCSPUP is set in the system prefix area if a user prefix is present in a journal record.

The journaled data then follows. If you want a length field for the data, you must include it in the data. Alternatively, you can compute the length of the data portion of a journal record by taking the length of the system header (10 bytes), plus the length of the system prefix (JCSPLL), plus the length of the user prefix (in the field, if any, defined by yourself), and subtracting the total from the length of the journal record (JCRLL):

```
JCRLL - (system header (10) bytes + JCSPLL + user prefix)
```

Not all journal records contain journaled data.

The CICS components that issue journaling requests are journal control, file control, FEPI, and terminal control.

```
*************************************************************************
* *            F U N C T I O N   I D E N T I F I E R S            * *
*************************************************************************
*                                                                       *
*        X'20' PLUS X'8-' ...USE FOR AUTOMATIC JOURNALING               *
*        X'40' PLUS X'8-' ...USE FOR AUTOMATIC LOGGING                  *
*************************************************************************
* *                         JOURNAL CONTROL                           * *
*************************************************************************
FIDJCLAB EQU   X'80'               ...JOURNAL CONTROL LABEL
*                                  RECORD (DFHJCR)                    *
*************************************************************************
* *                         FILE CONTROL                              * *
*************************************************************************
FIDALOG  EQU   X'40'               ...AUTOMATICALLY LOGGED
FIDAJRN  EQU   X'20'               ...AUTOMATICALLY JOURNALED
FIDMASS  EQU   X'10'               ...MASSINSERT REQ. (FIDFCWA ONLY)
*                                       PLUS ONE OF...                *
FIDFCRO  EQU   X'80'               ...FILE CONTROL READ-ONLY
FIDFCRU  EQU   X'81'               ...FILE CONTROL READ-UPDATE
FIDFCWU  EQU   X'82'               ...FILE CONTROL WRITE-UPDATE
FIDFCWA  EQU   X'83'               ...FILE CONTROL WRITE-ADD
FIDFCWAC EQU   X'84'               ...FILE CONTROL WRITE-ADD-COMPLETE
FIDFCWD  EQU   X'86'               ...FILE CONTROL WRITE DELETE
FIDFCBOF EQU   X'88'               ...BACKOUT FAILED LOG RECORD
FIDFCDSN EQU   X'8F'               ...DSNAME RECORD
*                                                                       *
*        NOTE THAT FID* VALUES (AS ABOVE) ARE OFTEN USED BOTH TO        *
*        IDENTIFY THE FUNCTION OF THE DWE AND THE FUNCTION OF THE       *
*        LOG RECORD.  IN THE CASE OF THE FIDFC* EQU'S ABOVE, THEY       *
*        ARE USED FOR LOG RECORDS ONLY.  THOSE BELOW APPLY ONLY         *
*        TO DWE'S                                                       *
*                                                                       *
FIDFCVWA EQU   X'80'               THIS DWE ADDRESSES A VSWA.
FIDFCRVY EQU   X'40'               THIS DWE IS ASSOCIATED WITH A       *
                                  RECOVERABLE CHANGE.
*************************************************************************
*               TERMINAL CONTROL FUNCTION IDENTIFIERS                   *
*                                                                       *
FIDTCML  EQU   X'F0'               SYNCPOINT - LOG SEQUENCE
*                                  NUMBERS                            *
*                                  CAN BE OR'ED WITH ANY OF           *
                                   THE FOLLOWING THREE FIELDS:
FIDTCDWL EQU   X'01'               ...DEFERRED WRITE DATA
FIDTCFMH EQU   X'02'               ...+ FUNCTION MANAGEMENT
*                                  HEADER
FIDTCDIP EQU   X'04'               ...+ DIP REQUEST
*                                                                       *
*        EQU   X'08'               ...DYNAMIC BACKOUT MASK            *
                                      RESERVED
FIDTCAL  EQU   X'40'               AUTOMATIC LOGGING MASK...
FIDTCAJ  EQU   X'20'               AUTOMATIC JOURNALING MASK..
*                          ...THE ABOVE 2 PLUS 1 OF FOLLOWING SET       *
FIDTCTL  EQU   X'80'               ...SEQUENCE NUMBER ONLY
*                                  (LOG ONLY)                         *
FIDTCIM  EQU   X'81'               ...INPUT MESSAGE (LOG AND
*                                  JOURNAL)                           *
FIDTCOM  EQU   X'82'               ...OUTPUT MESSAGE (JOURNAL
*                                  ONLY)                              *
FIDTCWP  EQU   X'83'               ...WRITE WAS PURGED (LOG
*                                  ONLY)                              *
FIDTCPRR EQU   X'84'               ...POSITIVE RESPONSE
*                                  RECEIVED (LOG ONLY)                *
FIDTCIMF EQU   X'85'               ...INPUT MESSAGE (W/FMH,
*                                  LOG AND JOURNAL)                   *
FIDTCOMN EQU   X'86'               ...OUTPUT MESSAGE, (W/O
*                                  FMH, JOURNAL ONLY)                 *
FIDTCON  EQU   X'87'               ...OUTPUT MESSAGE, FMH,
*                                  CCOMPL=NO                          *
FIDTCONN EQU   X'88'               ...OUTPUT MESSAGE, W/O FMH,
                                   ...CCOMPL=NO
FIDTCUA  EQU   X'89'               ...INITIAL TCT USER AREA
FIDTCEIB EQU   X'8A'                  INITIAL EXEC COMM AREA
```

```
*****************************************************************
* *              M O D U L E   I D E N T I F I E R S            * *
*****************************************************************
*                                                               *
MODIDTC  EQU   X'10'              ...TERMINAL CONTROL
MODIDFC  EQU   X'11'              ...FILE CONTROL
MODIDJC  EQU   X'45'              ...JOURNAL CONTROL
MODIDFEP EQU   X'50'              ...FEPI
*                                                               *
*****************************************************************
```

Figure 110. Journal module identifiers

Note:

1. Records created by automatic journaling and automatic logging are identified by values of X'20' (FIDAJRN) and X'40' (FIDALOG) respectively, added to the "base" value of the function identifier.
2. A File Control write-delete record created by the forward recovery process (where the write-delete is done) has a function identifier of X'86' (FIDFCWD). However, if the record is created by the autojournal process it has a function identifier of X'A2', made up of X'82' (write-update or FIDFCWU) plus X'20' (FIDAJRN).

Chapter 35. Identifying records for the start of tasks and UOWs

You can identify records written to mark the start of tasks by examining the value of the system prefix field JCSPF1. If the JCSPSOTK bit is set, the record has been written at the start of the task.

If the JCSPLSTK bit is set in field JCSPF1, then the record has been written at the start of the UOW.

Part 7. Format of journal records written to SMF

If you intend to write your own program to analyze the journaling records that are written to an SMF data set, you will need to know the format of the data.

The three components of the journaling record are an SMF block header, a CICS product section, and a CICS data section. The layout of an MVS SMF log, showing log blocks and CICS sections, is in Figure 111.

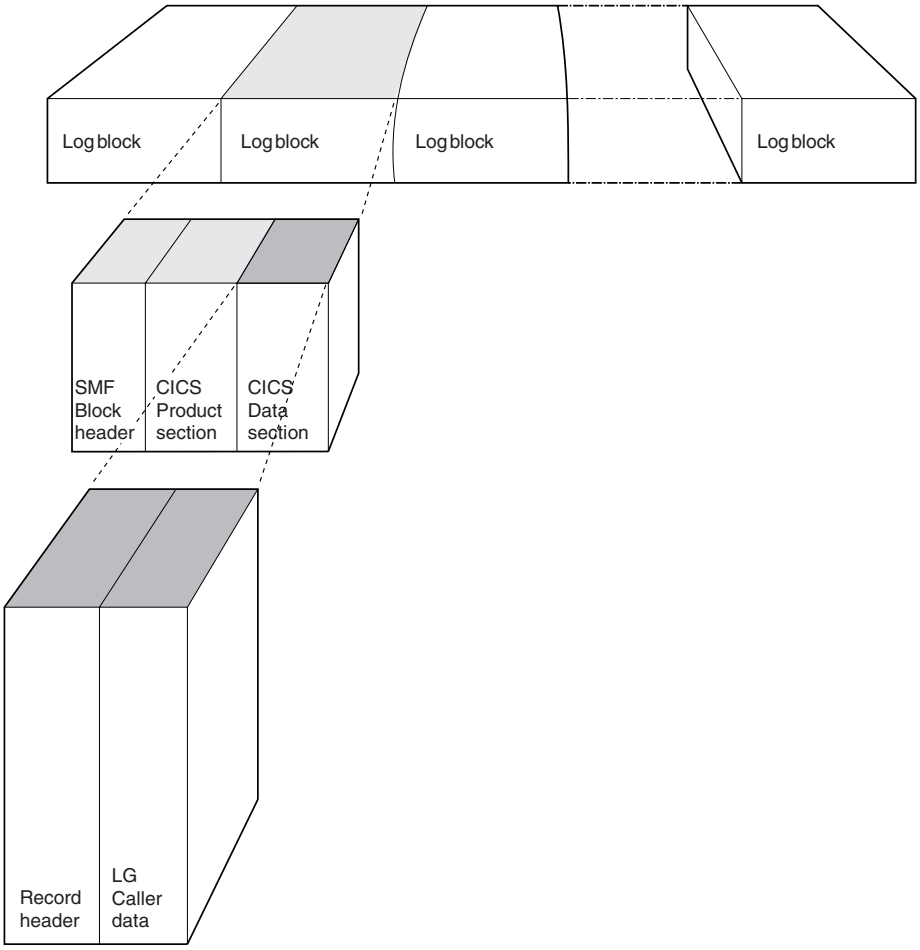


Figure 111. Layout of a CICS log written to MVS SMF

Journal records written to SMF can be read offline by user-written programs. Such programs can map journal records by including an INCLUDE DFHLGMSD statement. This generates the assembler version of the DSECT.

Chapter 36. The SMF block header

The SMF block header describes the system creating the output.

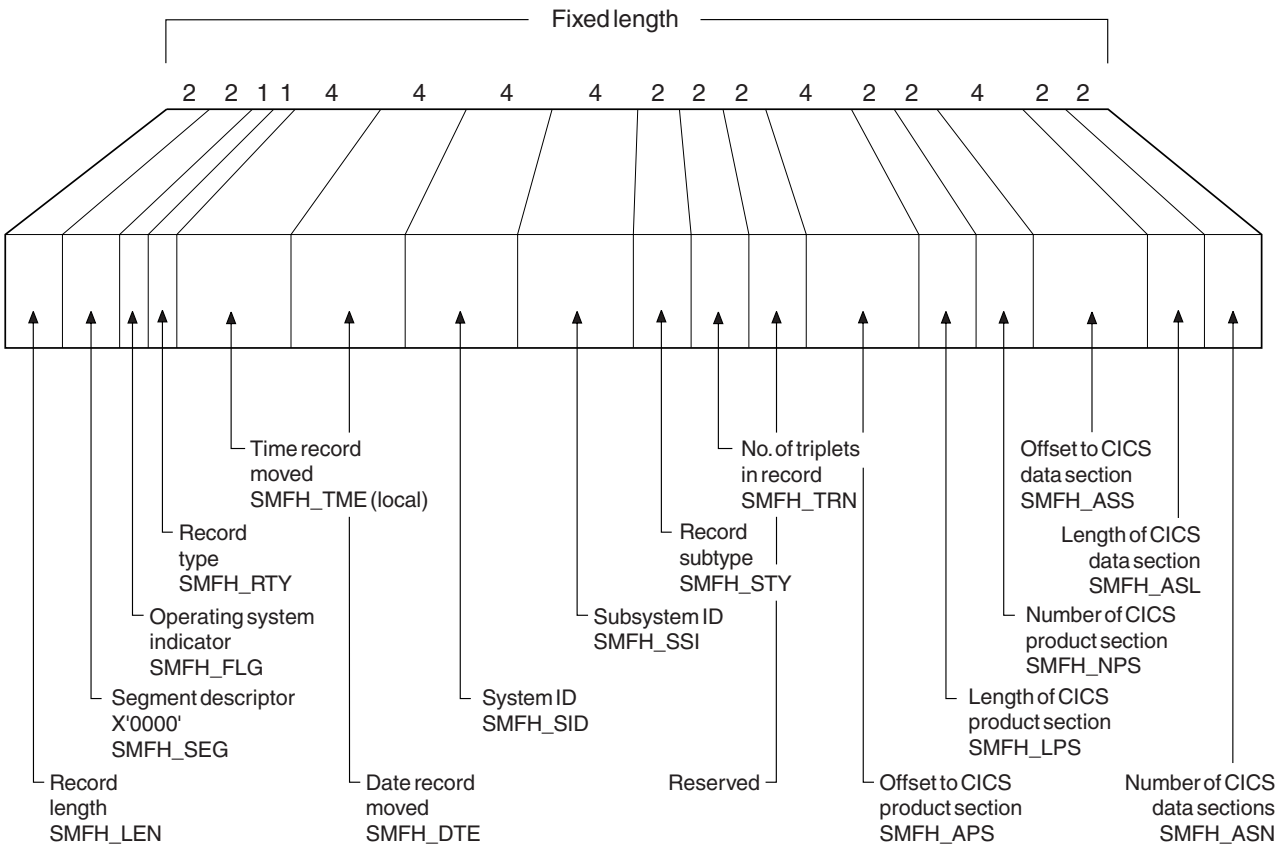


Figure 112. Format of the SMF block header

The format of the SMF block header is:

SMFH_LENGTH
2-byte record length.

SMFH_SEG
2-byte segment descriptor (X'0000').

SMFH_FLG
1-byte operating system indicator.

SMFH_RTY
1-byte record type.

SMFH_TME
4-byte (local) time record moved.

SMFH_DTE
4-byte date record moved.

SMFH_SID
4-byte system ID.

SMFH_SSI
4-byte subsystem ID.

SMFH_STY
2-byte record subtype.
SMFH_TRN
2-byte number of triplets in record.
Reserved
2-byte reserved field.
SMFH_APS
4-byte offset to CICS product section.
SMFH_LPS
2-byte length of CICS product section.
SMFH_NPS
2-byte number of CICS product section.
SMFH_ASS
4-byte offset to CICS data section.
SMFH_ASL
2-byte length of CICS data section.
SMFH_ASN
2-byte number of CICS data sections.

Note: CICS sets only the subsystem-related bits of the operating system indicator flag byte in the SMF header (SMFH_LG). SMF sets the remainder of the byte according to the operating system level and other factors. For an explanation of the setting of the other bits, refer to the *z/OS MVS System Management Facilities (SMF)* manual.

Chapter 37. The CICS product section

The CICS product section identifies the subsystem to which the journaling data relates.

Its format is shown in Figure 113.

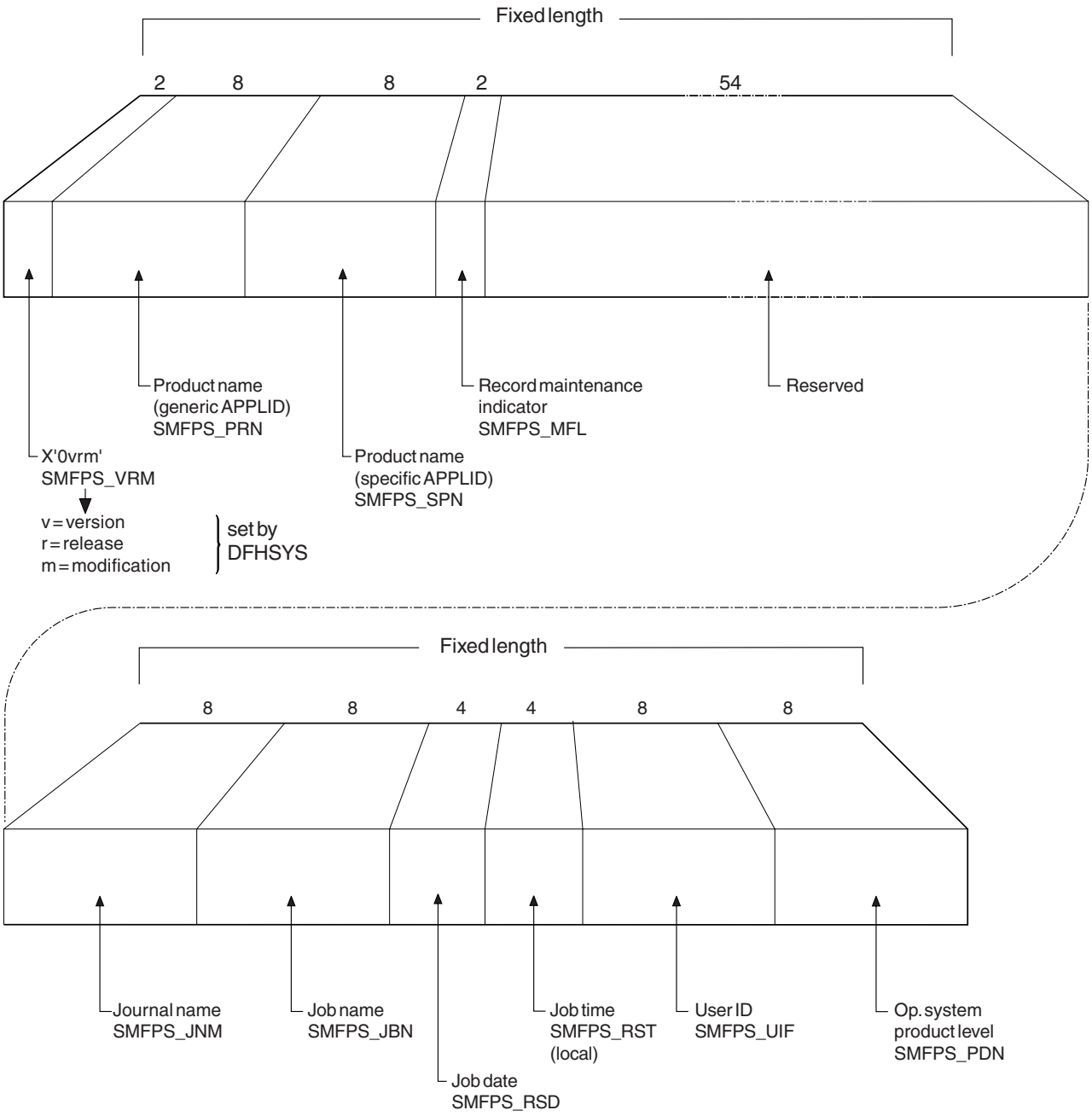


Figure 113. Format of the CICS product section

The format of the CICS product section is:

SMFPS_VRM
2-byte CICS version, release, and modification information, in the format X'0vrm'.
SMFPS_PRN
8-byte product name (generic APPLID).
SMFPS_SPN
8-byte product name (specific APPLID).
SMFPS_MFL
2-byte record maintenance indicator.
Reserved
54-byte reserved field.
SMFPS_JNM
8-byte journal name.
SMFPS_JBN
8-byte job name.
SMFPS_RSD
4-byte job date.
SMFPS_RST
4-byte job time (local).
SMFPS_UIF
8-byte user ID.
SMFPS_PDN
8-byte operating system product-level.

Chapter 38. The CICS data section

This section contains a variable number of CICS journal records. Each record comprises a general log record header, the format of which is shown in Figure 87 on page 667. This is followed by a user header, the format of which is shown in Figure 89 on page 671. This is then followed by the caller data.

If this is the first record being written to the journal after CICS initialization, the record comprises the general log record header, followed by a start-of-run record, the format of which is shown in Figure 88 on page 669. Subsequent records then take the form already described.

Part 8. Developing CICS compatibility interfaces

There are a number of ways that you can customize the dynamic allocation sample application program, used to allocate or deallocate data sets dynamically.

Chapter 39. Overview of the dynamic allocation program

The dynamic allocation (DYNALLOC) sample application program makes available to the CICS terminal operator most of the functions of DYNALLOC (SVC 99).

The DYNALLOC functions are described in z/OS MVS Programming: Authorized Assembler Services Reference (Volume 1). Functions that require authorized program facility (APF) authorization are not supported.

The application consists of one command-level assembler language program, DFH99, which is started by the transaction ADYN. The source code is provided in CICSTS52.CICS.SDFHSAMP.

Using DYNALLOC functions, the terminal operator can dynamically allocate or deallocate any data set that CICS opens and closes; for example, extrapartition transient data sets, journals, or dump and trace data sets.

Do *not* use the dynamic allocation program to allocate and deallocate data sets that are to be associated with files managed by file control; instead, use the dynamic allocation and deallocation facility that is part of CICS. If a file has not been allocated as part of CICS startup, CICS dynamic allocation occurs immediately before the file is opened, if sufficient information is in the resource definition for the file. The information needed is the data set name and disposition of the file. This information can be set by the **CEMT SET FILE** master terminal transaction, described in CEMT SET FILE in Reference > System definition, or the **EXEC CICS INQUIRE FILE** and **EXEC CICS SET FILE** commands, which provide additional inquiry and control facilities, and which are described in SET FILE in Reference > System programming.

To use the dynamic allocation sample program effectively, the terminal operator requires an understanding of the MVS job control language, or **TSO ALLOCATE** and **FREE** commands. For more information, including the error and reason codes returned by the DYNALLOC function, see z/OS MVS Programming: Authorized Assembler Services Reference (Volume 1).

The sample program uses a 3270 display screen terminal, and adjusts its formatting to suit the screen size. BMS is not required. The program is designed so that the installation can easily modify the functions supported to suit installation standards.

Chapter 40. Installing the program and transaction definitions

Transaction and program definitions for the dynamic allocation sample program are provided in the sample utilities resource definition group DFH$UTIL.

These definitions are installed using the CEDA command:

```
CEDA INSTALL GROUP(DFH$UTIL)
```

Note:

1. DFH99 **must** be defined with EXECKEY(CICS).
2. If you make any changes to the sample program, you must run the DFH99BLD procedure before using the ADYN transaction.

Chapter 41. The dynamic allocation program—terminal operation

When transaction ADYN is entered at a terminal, the operator is presented with a formatted display. The top part of the display is for entering commands, the bottom part for receiving messages from the program.

The operator types a command in TSO-like syntax, for example,

```
verb {keyword[(value...)]}...
```

and presses the ENTER key. The program checks the command for correct syntax, builds a DYNALLOC parameter list, and, if no serious errors are detected, issues a DYNALLOC SVC. Messages are then displayed to diagnose syntax errors, give the DYNALLOC return codes, and show any values returned by DYNALLOC information retrieval features. The command remains on the display, and the editing features of the terminal can be used to correct it for reentry, or to enter a different command.

If there are too many messages to fit into the message area of the screen, messages that cannot be displayed are queued, and the messages already on the screen are displayed with a brighter intensity to indicate that there are more messages to come. The operator can correct those errors that are being displayed, and reenter the command for further checking, when the queued messages, if any, are regenerated.

The program is terminated by entering a null command, which consists of pressing the ERASE INPUT key, followed by the ENTER key. PA keys 1 and 2 are ignored by the program. If you press the CLEAR key, you redisplay the last command entered. Pressing a program function key is equivalent to pressing ENTER.

Chapter 42. Using the dynamic allocation program's Help feature

The program includes a limited “help” feature, driven by the program's keyword table.

In response to “?”, the verb keywords are displayed. In response to “verb?”, all the operand keywords of that verb are displayed. For “verb operand(?)” a short description of the value expected for that operand is displayed. When a command containing “?” is entered, no DYNALLOC SVC is issued. “?” is recognized only in the positions specified above; the rest of the command is ignored.

Chapter 43. The dynamic allocation program—values

Values are classified as follows:

Keyword value
Keyword values must be specified for some keywords. For example, the STATUS keyword may have a keyword value of SHR, NEW, MOD, or OLD (which can be abbreviated).

String of key letters
The value can be a string of letters in any order. The program does not check that the combination of letters provided is meaningful. For example, for the RECFM keyword, the value can be a string of letters from A, B, D, F, G, M, R, S, T, U, and V.

Returned values
No value should be provided by the terminal operator, because this keyword requests a value to be returned by the DYNALLOC information retrieval features. The further description refers to the kind of value that will be returned. This is usually in the form in which the operator would enter it, although in a few cases the value is as a hexadecimal string.

Not allowed
Some keywords do not require a value, and you must not provide one.

Required
A value must be provided if the keyword coded is designated as requiring a value.

Optional
Specification of a value is optional for some keywords.

Single
Only one value may be provided for some keywords.

Multiple
For some keywords, more than one value is permitted. (In some cases, DYNALLOC requires more than one value, although the dynamic allocation sample program does not enforce this.)

Character string
Any characters are permitted in this type of value, although in most cases there will be additional rules to follow, for example, for the DSNAME keyword.

Numeric string
Only numeric characters are allowed for this type of value, for example, for the EXPDT keyword.

Maximum and minimum lengths
For character and numeric values, the maximum and minimum lengths of the value are checked by the program. For a fixed-length string, these values are the same. The value is still passed to DYNALLOC as specified.

Convertible to n byte binary
A numeric value is required, of a magnitude representable in binary in the specified number of bytes. Values that are too large are truncated to the maximum possible for the width.

The dynamic allocation sample program does not support negative numbers. It does not cross-check operand keywords; errors of this type usually cause DYNALLOC to return error codes of the form 03xx.

Abbreviation rules for keywords

Keywords can be abbreviated. A word in the command matches a keyword if:

- The spelling is the same.
- The first letter is the same, and the remaining letters in the word appear in the same order as they do in the keyword.

If an ambiguity occurs, the program diagnoses the ambiguity, and lists the possible keywords.

System programming considerations

Keyword spellings are defined in the program's table, DFH99T, which is link-edited with the program. Where possible, these are the same as the corresponding job control or TSO keywords. Comments in the source code for DFH99T explain how the system programmer can:

- Change the spelling of keywords
- Define alternative spelling for keywords
- Divide the functions of a verb into subsets
- Add new verbs with subset function
- Add new operands as they become available in the SVC.

Member DFH99BLD in CICSTS52.CICS.SDFHINST is the job stream used to build the program. If part of the program has been modified, reassemble that part and link-edit the program again.

The macros IEFZB4D0 (DYNALLOC parameter list structure) and IEFZB4D2 (symbolic key equates), provided by MVS, are used in the dynamic allocation program and its keyword table. The meaning of each keyword in the table is defined in terms of a symbolic name, defined by one of the macros IEFZB4D0 or IEFZB4D2. The definitions of command keywords given in that manual should be regarded in preference to those from any other source. To obtain a list of command keywords and their symbolic values, for use as a cross-reference to the MVS manual, assemble DFH99T with option SYSPARM(LIST), and print the resulting object code. If the table is changed, repeat the assembly to obtain a new list.

Chapter 44. The flow of control when a DYNALLOC request is issued

When a DYNALLOC request is issued, the main program, DFH99M, calls a series of subsidiary programs to process the request.

The flow in a normal invocation is as follows. The main program, DFH99M, receives control from CICS and carries out initialization. This includes determining the screen size, allocating input and output buffer sections, and issuing initial messages. It then invokes DFH99GI to get the input command from the terminal. Upon return, if the command was null, the main program terminates, issuing a final message.

The command obtained has its start and end addresses stored in the global communication area, COMM. The main program allocates storage for tokenized text, and calls DFH99TK to tokenize the command. If errors were detected at this stage, further analysis of the command is bypassed.

Following successful tokenizing, the main program calls DFH99FP to analyze the verb keyword. DFH99FP calls DFH99LK to look up the verb keyword in the table, DFH99T. DFH99LK calls DFH99MT if an abbreviation is possible. Upon finding the matching verb, DFH99FP puts the address of the operand section of the table into COMM, and puts the function code into the DYNALLOC request block.

The main program now calls DFH99KO to process the operand keywords. Each keyword in turn is looked up in the table by calling DFH99LK, and the value coded for the keyword is checked against the attributes in the table. DFH99KO then starts off a text unit with the appropriate code and, depending on the attributes the value should have, calls a conversion routine.

For character and numeric strings, DFH99CC is called. It validates the string, and puts its length and value into the text unit.

For binary variables, DFH99BC is called. It validates the value, converts it to binary of the required length, and puts its length and value into the text unit.

For keyword values, DFH99KC is called. It looks up the value in the description part of the keyword table using DFH99LK, and puts the coded equivalent value and its length into the text unit.

When a keyword specifying a returned value is encountered, DHF99KO makes an entry on the returned value chain, which is anchored in COMM. This addresses the keyword entry in DFH99T, the text unit where the value is returned, and the next entry. In this case the conversion routine is still called, but it only reserves storage in the text unit, setting the length to the maximum and the value to zeros.

When all the operand keywords have been processed, DFH99KO returns to the main program, which calls DFH99DY to issue the DYNALLOC request.

DFH99DY sets up the remaining parts of the parameter list and, if no errors too severe have been detected, a subtask is attached to issue the DYNALLOC SVC. A WAIT EVENT is then issued against the subtask termination ECB. When the

subtask ends and CICS dispatches the program again, the DYNALLOC return code is captured from the subtask ECB and the error and reason codes from the DYNALLOC request block and a message is issued to give these values to the terminal.

DFH99DY then returns to the main program, which calls DFH99RP to process returned values. DFH99RP scans the returned value chain, and for each element issues a message containing the keyword and the value found in the text unit. If a returned value corresponds to a keyword value, DFH99KR is called to look up the value in the description, and issue the message.

Processing of the command is now complete, and the main program is reinitialized for the next one, and loops back to the point where it calls DFH99GI.

Messages are issued at many places, using macros. The macro expansion ends with a call to DFH99MP, which ensures that a new line is started for each new message, and calls DFH99ML, the message editor. Input to the message editor is a list of tokens, and each one is picked up in turn and converted to displayable text. For each piece of text, DFH99TX is called, which inserts the text into the output buffer, starting a new line if necessary. This ensures that a word is never split over two lines.

At the end of processing the command, the main program calls DFH99MP with no parameters, which causes it to send the output buffer to the terminal, and initialize it to empty.

Part 9. Customizing security processing

You can customize security processing by invoking a security manager other than RACF, and by writing a "good night" transaction. You can also use SAML.

Authorization routines

In z/OS, do not install SVCs or PC routines that return control to their caller in any authorized mode: that is, in supervisor state, system PSW key, or APF-authorized. Doing so is contrary to the z/OS Statement of Integrity (http://www.ibm.com/systems/z/os/zos/features/racf/zos_integrity_statement.html).

If you invoke such services from CICS, you might compromise your system integrity, and any resultant problems will not be resolved by IBM Service.

Chapter 45. Invoking an external security manager

CICS provides an interface to an external security manager (ESM), which can be the Resource Access Control Facility (RACF), a vendor product, or user-written. This information provides an overview of the CICS-ESM interface, and describes how you can use the MVS router exit to pass control to a user-written ESM. It describes how ESM exit programs can access CICS-related information. Finally, it lists the control points at which CICS invokes the ESM.

Note: This information is intended primarily for non-RACF users. For definitive information about security processing using RACF, refer to RACF facilities in Securing.

An overview of the CICS-ESM interface

CICS security uses, the MVS system authorization facility (SAF) interface to route authorization requests to the external security manager (ESM). Typically, if RACF is present, the MVS router passes control to it. However, you can modify the action of the MVS router by invoking the router exit.

The router exit can be used, for example, to pass control to a user-written or vendor-supplied ESM. If you want to use your own security manager, you must supply an MVS router exit routine.

The MVS router

The system authorization facility (SAF) provides your installation with centralized control over security processing by using a system service called the *MVS router*. The MVS router provides a common system interface for all products providing and requesting resource control.

The resource-managing components and subsystems (such as CICS) call the MVS router as part of certain decision-making functions in their processing, for example access control checking and authorization-related checking. These functions are called *control points*. This single SAF interface encourages the use of common control functions shared across products and across systems.

If RACF is available in the system, the MVS router might pass control to the RACF router, which in turn invokes the appropriate RACF function. The parameter information and the RACF router table, which associates router invocations with RACF functions, determine the appropriate function. However, before calling the RACF router, the MVS router calls an optional installation-supplied security-processing exit, if one has been installed.

The system authorization facility and the SAF router are present on all MVS systems, even if RACF is not installed. Although the SAF router is not part of RACF, many system components and programs, such as CICS, invoke RACF through the RACROUTE macro and SAF. Therefore, installations can modify RACF parameter lists and do customized security processing within the SAF router. For information about how to code a SAF router exit, see z/OS Security Server RACF Messages and Codes.

For more information about the MVS router, see System authorization facility (SAF) in z/OS Security Server RACROUTE Macro Reference and ICHRTX00 - MVS Router Exit in z/OS MVS Installation Exits.

Passing control to a user-supplied ESM

Usually, a caller (such as CICSPlex SM) invokes the MVS router and passes it request type, requester, and subsystem parameters through the RACROUTE exit parameter list. The MVS router uses these parameters and calls the router exit. When the router exit completes its processing, it passes a return code to the router.

If the return code is 0, the router invokes RACF. RACF reports the result of that invocation to the router by entering return and reason codes in register 15 and register 0, respectively. The router converts the RACF return and reason codes to router return and reason codes and passes them to the caller. The router provides additional information to the caller by placing the unconverted RACF return and reason codes in the first and second words of the router input parameter list.

If your installation does not use RACF, you can make the MVS router exit pass control to an alternative ESM. However, if you do so you must still provide CICSPlex SM with the RACF return and reason codes that it expects to receive. You set the router exit return code so that RACF is not invoked; and you simulate the results of a RACF invocation by coding the exit so that it places the RACF return and reason codes in the first and second fullwords of the router input parameter list. RACF return and reason codes are documented in the *z/OS MVS Programming: Assembler Services Reference*.

For more information about passing control to a user-supplied ESM, see Simulating a Call to RACF in z/OS MVS Installation Exits.

Using ESM exit programs to access CICS-related information

When CICS invokes the ESM, it passes information about the current CICS environment, for use by an ESM exit program, in an **installation data parameter list**. How your exit programs access the installation data parameter list depends on whether or not your ESM is RACF.

For non-RACF users — the ESM parameter list

CICS (or another caller) passes information to your external security manager in the ESM parameter list, the address of which can be calculated using field SAFPRACP of the MVS router parameter list.

When the caller is CICS, the “INSTLN” field of the ESM parameter list points to the installation data parameter list, which contains CICS-related information that can be used by ESM exit programs.

The format of the ESM parameter list, and the actual name of the “INSTLN” field, vary, depending on which CICS security event is being processed. (The “request type” field (SAFPREQT) of the router parameter list shows why the ESM is being called by indicating the RACROUTE REQUEST type.) Table 38 on page 723 shows how some formats of the ESM parameter list can be mapped using MVS macros.

Table 38. Mapping the ESM parameter list

RACROUTE REQUEST type	Parameter list mapping macro	INSTLN field name
VERIFY	IRRPRIPL	INITIPTR (X'10')
AUTH	ICHACHKL	ACHKIN31 (X'20')
FASTAUTH	Not available	Offset X'18'
LIST	Not available	Offset X'0C'
EXTRACT	Not available	None

Note: The INSTLN field points to the installation parameter list only if you specify INSTLN on the ESMEXITS system initialization parameter. The default value of this parameter is NOINSTLN, which means that no installation data is passed.

For RACF users — the RACF user exit parameter list

If you are a RACF user, you can find the address of the installation data parameter list directly from the RACF user exit parameter list. The name of the relevant field in the user exit parameter list varies according to the RACROUTE REQUEST type and the RACF user exit that is invoked.

The relationships between REQUEST type, exit name, and field name are shown in Table 39.

Table 39. Obtaining the address of the installation data parameter list

RACROUTE REQUEST type	RACF exit	Exit list mapping macro	Parameter list field name
VERIFY	ICHRIX01	ICHRIXP	RIXINSTL
VERIFY	ICHRIX02	ICHRIXP	RIXINSTL
AUTH	ICHRCX01	ICHRCXP	RCXINSTL
AUTH	ICHRCX02	ICHRCXP	RCXINSTL
FASTAUTH	ICHRFX01	ICHRFXP	RFXANSTL
FASTAUTH	ICHRFX02	ICHRFXP	RFXANSTL
LIST	ICHRLX01	ICHRLX1P	RLX1INST
LIST	ICHRLX02	ICHRLX2P	RLX2PRPA See note 2.
EXTRACT	Not available	Not available	None

Note:

1. The *xxx*INSTL field points to the installation parameter list only if you specify INSTLN on the ESMEXITS system initialization parameter. The default value of this parameter is NOINSTLN, which means that no installation data is passed.
2. RLX2PRPA contains the address of the ICHRLX01 user exit parameter list (RLX1P). Field RLX1INST of RLX1P in turn points to the installation data parameter list.

For full descriptions of the RACF exit parameter lists, see z/OS Security Server RACF Security Administrator's Guide. For more information about CICS security processing using RACF, see RACF facilities in Securing.

The installation data parameter list

The installation data parameter list gives your ESM exit programs access to the following information:

- The CICS security event being processed.
- Details of the current CICS environment. That is:
  - The applid of the CICS region
  - The common work area (CWA)
  - The transaction being invoked
  - The program being executed
  - The CICS terminal identifier
  - The SNA LU name
  - The terminal user area.

You can map the installation parameter list using the macro DFHXSUXP. The DSECT DFHXSUXP contains the following fields:

UXPLEN
A halfword containing the length of this parameter list in bytes.

UXPARROW
Arrow “eyecatcher” (>).

UXPDFHXS
The name of the owning component (DFHXS).

UXPBLKID
The name of the block identifier (UXPARMS).

UXPPHASE
Address of a 1-byte code that indicates the reason for the call to the ESM (that is, the security event being processed). The code can have one of the following values:

DEFAULT_SIGN_ON (X'01')
Signon of default userid

PRESET_SIGN_ON (X'02')
Signon of preset security terminal

IRC_SIGN_ON (X'03')
Link signon for IRC (MRO) links

LU61_SIGN_ON (X'04')
Link signon for LUTYPE6.1 links

LU62_SIGN_ON (X'05')
Link signon for APPC links

XRF_SIGN_ON (X'06')
XRF tracking of signon

ATTACH_SIGN_ON (X'07')
Attach-time signon of link user

NON_TERMINAL_SIGN_ON (X'08')
Signon of a non-terminal userid

USER_SIGN_ON (X'10')
Normal user signon

PRESET_SIGN_OFF (X'22')
Sign-off when terminal deleted

LINK_SIGN_OFF (X'25')
Sign-off when link is closed

XRF_SIGN_OFF (X'26')
XRF tracking of sign-off

ATTACH_SIGN_OFF (X'27')
End-of-task sign-off of link user

NON_TERMINAL_SIGN_OFF (X'28')
Sign-off of a non-terminal userid

USER_SIGN_OFF (X'30')
Normal user sign-off

TIMEOUT_SIGN_OFF (X'31')
Sign-off forced by the terminal abnormal condition program, or time-out by the CSSC transaction

USRDELAY_SIGN_OFF (X'32')
Sign-off caused by expiry of USRDELAY interval

DEFERRED_SIGN_OFF (X'33')
Sign-off deferred to task end

USER_ATTACH_CHECK (X'40')
Transaction attach check for user

LINK_ATTACH_CHECK (X'41')
Transaction attach check for link

EDF_ATTACH_CHECK (X'42')
Transaction attach check for CEDF

USER_COMMAND_CHECK (X'50')
Command checking for user

LINK_COMMAND_CHECK (X'51')
Command checking for link

EDF_COMMAND_CHECK (X'52')
Command checking for EDF

USER_RESOURCE_CHECK (X'60')
Resource checking for user

LINK_RESOURCE_CHECK (X'61')
Resource checking for link

EDF_RESOURCE_CHECK (X'62')
Resource checking for EDF

USER_SURROGATE_CHECK (X'68')
Surrogate checking for user

LINK_SURROGATE_CHECK (X'69')
Surrogate checking for link

EDF_SURROGATE_CHECK (X'6A')
Surrogate checking for EDF

USER_QUERY_CHECK (X'70')
Query checking for user

LINK_QUERY_CHECK (X'71')
Query checking for link

EDF_QUERY_CHECK (X'72')
Query checking for EDF

INITIALIZE_SECURITY (X'80')
Initialization of CICS security

REBUILD_SECURITY (X'81')
CEMT or command-level SECURITY REBUILD

XRF_TRACK_INITIALIZE (X'82')
XRF tracking of initial or rebuild.

UXPSUBSY
Address of an area containing the CICS subsystem identifier.

UXPAPPL
Address of an area containing the CICS application ID.

Note: When CICS is a member of a z/OS Communications Server for SNA generic resource, the area pointed to by UXPAPPL contains the *generic*, not the specific, applid.

UXPCWA
Address of the Common Work Area.

UXPTRAN
Address of an area containing the transaction identifier.

UXPPROG
Address of an area containing the program name. The address may be zero if no program name can be identified.

UXPTERM
Address of an area containing the terminal identifier. The address may be zero if no terminal is associated with the request.

UXPLUNAM
Address of an area containing the SNA LU name. The address may be zero if no terminal is associated with the request, or the area may be blank if the terminal is not an SNA terminal.

UXPTCTUA
Address of the TCT user area.

UXPTCTUL
Address of a fullword containing the length of the TCTUA.

UXPCOMM
Address of a 2-word communication area.

Using early verification processing

The CICS signon routine invokes the SAF interface, using the RACROUTE REQUEST=VERIFY macro with the ENVIR=VERIFY option in problem-program state. Invoking this version of the macro has no effect if the ESM is RACF, but other external security manager products can get control through the SAF exit interface, and perform their own *early verification* routine.

CICS defers the creation of the accessor environment element until the RACROUTE REQUEST=VERIFY macro with the ENVIR=CREATE option is issued

to perform the *normal verification* routine. The ENVIR=CREATE version of the macro is issued by the security manager domain running in supervisor state.

CICS passes the following information on the ENVIR=VERIFY version of the RACROUTE REQUEST=VERIFY macro:

USERID
The userid of the user signing on to the CICS region.

GROUP
The group name, if specified, of the group into which the user wants to sign on.

PASSWRD
The user's password to verify the userid.

NEWPASS
A new value, if specified, for the user's password. This changes the existing password and is to be used for subsequent signons.

OIDCARD
The contents, if supplied, of an operator identification card.

APPL
The APPLID of the CICS region on which the user is signing on. Which APPLID is passed depends on what is specified as the system initialization parameter.

INSTLN
A pointer to a vector of CICS-related information, which you can map using the DFHXSUXP mapping macro. This pointer is valid only if ESMEXITS=INSTLN is specified as a system initialization parameter for the CICS region.

The installation data referenced by the INSTLN parameter includes a pointer, UXPCOMM, to a two-word communications area that can be used to pass information between the two phases of the signon verification process—between the early verification routine initiated by ENVIR=VERIFY, and the normal verification routine initiated by ENVIR=CREATE.

CICS maintains a separate communications area for each task, in CICS-key storage.

Writing an early verification routine

An early verification routine, written for the `ENVIR=VERIFY` option, receives control from SAF in the usual way from the external security manager whose entry point is addressed by field SAFVRACR in the SAF vector table.

It receives control in the same state as its caller, as follows:

- Problem-program state
- Task mode (usually the CICS quasi-reentrant TCB)
- PSW storage key 8
- 31-bit addressing mode
- Primary address translation mode.

Register 13 points to a standard 18-word save area. Register 1 points to a 2-word parameter list, where:

- The first word is the address of the SAF parameter list for the VERIFY function.

- The second word is the address of a 152-byte work area.

Using CICS API commands in an early verification routine

An early verification routine can use CICS application programming interface (API) commands, provided it obeys the following interface rules:

- The routine must be written in assembler.
- Entry to the routine must be via the DFHEIENT macro, which saves the caller's registers and establishes a CICS early verification API environment.
- Exit from the routine must be via the DFHEIRET macro, which releases the CICS early verification API environment and restores the caller's registers.
- The routine *must* be link-edited with the special security domain API stub, DFHXSEAI, *instead of* the normal CICS API stub, DFHEAI0. The CICS early verification stub causes linkage to a special interface routine that is aware of the SAF interface linkage requirements, and saves the current CICS command environment. In addition, the standard EXEC interface stub DFHEAI should also be included, immediately before the early verification routine, with an ORDER statement:

```
INCLUDE SYSLIB(DFHXSEAI)
INCLUDE SYSLIB(DFHEAI)
ORDER   DFHEAI,verify-program,DFHEAI0
ENTRY   verify-program
```

The DFHEIENT and DFHEIRET macros are inserted by the CICS translator unless you specify

```
*ASM XOPTS(NOPROLOG,NOEPILOG)
```

as the first statement of the program. The DFHEIENT macro assumes that register 15 points to its first executable instruction.

Upon return from the DFHEIENT macro, a CICS storage area mapped by the DFHEISTG macro has been established. The pointer DFHEIBP (and the register specified in the EIBREG parameter of DFHEIENT) contains the address of an EXEC interface block (EIB). DFHEICAP contains the pointer to the original parameter list supplied by the SAF interface.

Return and reason codes from the early verification routine

Before returning control, the early verification routine should set a return code and reason code in fields SAFPRRET and SAFPRREA of the SAF parameter list. It should also pass a value to be returned as the SAF return code in a register that is specified in the RCREG keyword of the DFHEIRET macro that is used to exit the program.

These return codes are examined by the CICS signon function, and any non-zero value in SAFPRRET is interpreted as a verification failure and causes the signon to fail. A zero return code allows the signon to proceed, and eventually CICS issues a `RACROUTE REQUEST=VERIFY,ENVIR=CREATE` macro in supervisor state and under control of the CICS resource-owning TCB. It is only at this invocation that CICS accepts an ACEE address from the external security manager.

Chapter 46. Writing a “good night” program

You can use the **GNTRAN** system initialization parameter to specify a “good night” transaction that you want CICS to invoke when a terminal times out.

The default value for **GNTRAN** is 'NO', which means that CICS does not schedule a “good night” transaction, but instead tries to sign off the terminal user. Whether or not the sign off is successful depends on the value of the SIGNOFF attribute on the TYPETERM definition of the terminal.

Any transaction that you specify on the **GNTRAN** parameter must be able to handle the type of communication area it is passed when terminal timeout occurs. The CICS sign-off transaction, CESF, can do this, but CESN and all other supplied transactions cannot. For further information, see GNTRAN system initialization parameter in Reference -> System definition.

Writing your own “good night” program allows you to include functions in addition to, or instead of, sign-off. For example, your program could prompt the terminal user to enter their password, and allow the session to continue if the correct response is received. CICS supplies a sample “good night” program, DFH0GNIT, that demonstrates this, and a sample TRANSACTION resource, GNIT, that points to DFH0GNIT.

CICS passes the “good night” program a parameter list in the communications area shown in Figure 114 on page 730. If a terminal times out during a pseudoconversational transaction, your program can perform the following actions, using information in the parameter list:

- Ask for and check a response from the user
- Restore the screen left by the timed-out transaction
- Restore the cursor position
- Receive the communications area of the timed-out transaction, which is passed to the “good night” transaction as an input message
- Return with the TRANSID of the next transaction in the conversation.

The communications area of the “good night” program

Figure 114 on page 730 shows the communications area passed to the “good night” program.

```
DFHSNGS
DFHSNGS_FIXED                 DS    0CL64    Fixed part of parameter list

GNTRAN_START_TRANSID          DS    CL4      TRANSID that invoked GNTRAN
GNTRAN_PSEUDO_CONV_FLAG       DS    CL1      Pseudoconversational flag
GNTRAN_SCREEN_TRUNCATED       DS    CL1      Screen buffer truncation flag
GNTRAN_TRANSLATE_TIOA         DS    CL1      Uppercase translation flag
                              DS    CL9      Reserved
GNTRAN_TIMEOUT_TIME           DS    CL8      Time of terminal timeout
GNTRAN_TIMEOUT_REASON         DS    CL1      Reason for timeout
                              DS    CL11     Reserved
GNTRAN_PSEUDO_CONV_TRANSID    DS    CL4      Next transaction ID
GNTRAN_SCREEN_LENGTH          DS    FL2      Length of screen buffer
GNTRAN_CURSOR_POSITION        DS    FL2      Cursor position
GNTRAN_SCREEN_WIDTH           DS    FL2      Width of screen
GNTRAN_SCREEN_HEIGHT          DS    FL2      Height of screen
GNTRAN_USER_FIELD             DS    CL16     Available to user program
DFHSNGS_VARIABLE              DS    0X       Variable part of parameter list
GNTRAN_SCREEN_BUFFER          DS    0X       Contents of screen buffer
```

Figure 114. Communications area passed to the "good night" program (assembler)

GNTRAN_START_TRANSID
The identifier of the transaction that started the "good night" transaction. If it was started by CICS because of a terminal timeout, GNTRAN_START_TRANSID is set to 'CEGN'. Your program should examine this field to check that timeout processing is appropriate (that is, that the "good night" transaction was started because of a terminal timeout and for no other reason).

GNTRAN_PSEUDO_CONV_FLAG
A flag indicating whether the terminal timed out during a pseudoconversational transaction.

Y The terminal timed out between transactions that form part of a pseudoconversational application.

N The terminal did not time out between transactions that form part of a pseudoconversational application.

GNTRAN_SCREEN_TRUNCATED
A flag indicating whether the 3270 screen buffer had to be truncated.

Y The screen buffer was truncated.

N The screen buffer was not truncated.

GNTRAN_TRANSLATE_TIOA
An internal flag indicating whether DFHZSUP is to translate the TIOA to uppercase, if required by the TYPETERM or PROFILE setting:

Y The TIOA is to be translated.

N Uppercase translation is to be bypassed.

GNTRAN_TIMEOUT_TIME
The time that the terminal timed out, in CICS ABSTIME format.

GNTRAN_TIMEOUT_REASON
The reason for the timeout:

T No input from the terminal

X An XRF takeover.

GNTRAN_PSEUDO_CONV_TRANSID
The identifier of the next transaction, if the terminal timed out during a

pseudoconversational sequence. (If the terminal did *not* time out during a pseudoconversational sequence, the value of this field is meaningless.)

GNTRAN_SCREEN_LENGTH
The length of the screen buffer.

GNTRAN_CURSOR_POSITION
The cursor position.

GNTRAN_SCREEN_WIDTH
The width of the screen in use when the terminal timed out.

GNTRAN_SCREEN_HEIGHT
The height of the screen in use when the terminal timed out.

You can use GNTRAN_SCREEN_WIDTH and GNTRAN_SCREEN_HEIGHT to decide whether to use the ERASE DEFAULT or ERASE ALTERNATE option when restoring the user's screen.

GNTRAN_USER_FIELD
This field is available for use by your "good night" user program. It is initialized to binary zeroes and is not changed by CICS. You can use it to help develop a pseudoconversational "good night" transaction.

GNTRAN_SCREEN_BUFFER
A variable length field containing the contents of the screen buffer.

The sample "good night" program, DFH0GNIT

The sample "good night" program is a pseudoconversational COBOL program named DFH0GNIT.

Copy books of the communications area passed to the "good night" program are supplied in assembler language, COBOL, PL/I, and C. The names of the supplied program, copy books, and mapset, and the CICSTS52.CICS libraries in which they can be found, are summarized in Table 40.

Table 40. Sample "good night" program, copy books, and mapset

Language	Member name	Library
Program source: COBOL only	DFH0GNIT	SDFHSAMP
Copy books: Assembler COBOL PL/I C	DFHSNGSD DFHSNGSO DFHSNGSL DFHSNGSH	SDFHMAC SDFHCOB SDFHPL1 SDFHC370
Mapset:	DFH$GMAP	SDFHSAMP

What the sample program does

The DFH0GNIT sample program:

1. Checks that it has been invoked for a terminal timeout, by testing the GNTRAN_START_TRANSID field of the communications area passed by CICS. If this contains anything other than 'CEGN', it quits.
2. If a flag within GNTRAN_USER_FIELD shows that this is the first invocation for this timeout:
   a. If GNTRAN_PSEUDO_CONV_FLAG indicates that the terminal timed out during a pseudoconversation, issues EXEC CICS RECEIVE to retrieve the communications area.

b. Saves the length of the communications area in another field within GNTRAN_USER_FIELD.
c. Writes the communication area, if any, to a temporary storage queue.
d. Displays a screen asking the user to input his or her password, and sets the flag indicating that this has been done.
e. Issues EXEC CICS RETURN with TRANSID GNIT and the COMMAREA option, to continue the timeout process as a pseudoconversation.

3. If this is **not** the first invocation for this timeout:
   a. Recovers the original communication area, if any, from the temporary storage queue.
   b. Checks the password received from the user, and redisplays the timeout screen with an error message if it is incorrect.
4. If the number of incorrect responses exceeds the maximum specified to your external security manager, DFH0GNIT returns immediately with TRANSID CESF, which tries to sign off the userid.
5. If the correct password is entered, DFH0GNIT:
   - Restores the screen contents
   - Restores the cursor position.

   If the terminal timed out during a pseudoconversational transaction, DFH0GNIT also:
   - Restores the communications area of the timed-out transaction
   - Returns with the TRANSID of the next transaction in the interrupted conversation.

Customizing the sample "good night" program

You can write your "good night" program in any of the languages supported by CICS, with full access to the CICS application and system programming interfaces.

If you customize the supplied program, or write your own "good night" program, note the following:

- Like the sample, your program should be pseudoconversational, because it could be invoked simultaneously for many users (if, for example, many terminals time out during the lunch period). If your program is conversational, CICS maximum number of tasks (MXT) could quickly be reached.

  When you are continuing your timeout program's pseudoconversation, always specify the name of your "good night" transaction (for example, GNIT) as the next TRANSID. If you do not, CICS does not know that you are still handling the timeout, and results may be unpredictable.
- Your program should always start, like the sample program, by testing the GNTRAN_START_TRANSID field of the communications area passed by CICS. If it finds that the "good night" transaction was started for any reason other than a terminal timeout (for example, by an EXEC CICS START request), timeout processing may not be appropriate.
- To obtain the communications area of the timed-out transaction in a pseudoconversation, your program must issue an EXEC CICS RECEIVE command. (The communication area passed to it on invocation is **not** that of the timed-out transaction, but contains information about the timed-out transaction.)
- If your program tries to sign off the terminal user, the result depends on what is specified on the SIGNOFF option of the terminal's TYPETERM definition:

  YES The terminal is signed off, but not logged off.

NO The terminal remains logged on and signed on.

LOGOFF
The terminal is both signed off and logged off.

- Specify the identifier (TRANSID) of your "good night" transaction on the GNTRAN system initialization parameter.

  If you have customized the sample program, DFH0GNIT, specify the supplied sample transaction definition, GNIT.

  If you have written your own "good night" program, named something other than DFH0GNIT, you must create and install a transaction definition that points to your program, and specify this definition on the GNTRAN SIT parameter.

Chapter 47. Security processing using SAML

CICS supports Security Assertion Markup Language (SAML) by providing an application programming interface (API), which consists of a linkable interface DFHSAML, a channel, and a set of containers.

Pattern: logging user information

A typical pattern is for a SAML-aware application to write information about the user to a log (for example, by using WRITE JOURNALNAME). The information that is written can include the DFHSAML-NAMEID and related containers. Including this information allows the request to be audited together with the user, even though the request itself might be running under a common user ID.

For more information about SAML support, see Overview of SAML support.

Restrictions

Use this reference material to understand SAML functions that are not supported by CICS.

- CICS supports the SAMLCore1.1 and SAMLCore2.0 standards. It does not support the protocols that are described in those standards. It does not support SAMLCore 1.0.
- Do not use the Inclusive XML Canonicalization algorithm for signed tokens, because this might cause the signature verification to fail. Make sure your Identity Provider does not specify any InclusiveNamespaces list, which lists the element namespaces that are handled in the manner described by the Inclusive XML Canonicalization.
- CICS supports only embedded signatures.
- CICS does not check the version of an assertion. To enforce the rules specified in the SAML specifications, you must implement that behaviour in your application.
- CICS does not support ProxyRestriction in a SAML assertion.
- CICS does not support multiple SubjectConfirmation methods specified in the SAML assertion. If multiple SubjectConfirmation methods are specified, CICS returns the last one.
- The container DFHSAML-SUBJADDR is not supported for SAML 2.0.
- CICS SAML support supports only the RSA algorithm for signing tokens and for signature verification.
- CICS SAML support does not support DSA (Digital Signature Algorithm) keys. If your keyring contains a certificate with a DSA key, the keyring cannot be opened and SAML processing does not work. SAML uses the same keyring as SSL and Web Services Security. If you use DSA keys with either SSL or Web Services Security, you must use a different region for SAML processing.
- SAML processing does not work if the keyring contains a site certificate or a certificate that is owned by a user ID that does not have a private key. If you want to have such certificates on your keyring, flag them as `USAGE(CERTAUTH)`.
- CICS does not support X509SubjectName or X509SubjectIssuer in SAML assertion signatures. If you specify either of these elements in the signature X509Data, the signature verification fails.

- The use of certificates with the same label in a keyring is not supported.
- Personal certificates belonging to users other than CICS that are connected to the SAML keyring are not supported.

Part 10. Customizing resource definition operations with user-written programs

You can customize the behavior of the CEDA transaction and the DFHCSDUP utility program used for working with resource definitions on the CSD.

- You can invoke RDO functions from an application program by linking to program DFHEDAP.
- You can invoke a user program from DFHCSDUP or invoke DFHCSDUP from a user program
- You can modify the behavior of DFHCSDUP by passing control to an exit routine at key points in the program's processing.

A general note about user-written programs

The following comment applies to all user-written programs mentioned in Part 8 of this book:

- Upon return from any customer-written program, CICS must always receive control in primary-space translation mode, with the original contents of all access registers restored, and with all general purpose registers restored (except for those which provide return codes or linkage information).

  For information about translation modes, refer to the *IBM z/Architecture Principles of Operation* manual.

Chapter 48. Using the programmable interface to CEDA

The resource definition online (RDO) transaction, CEDA has a programmable interface that you can use to invoke the functions provided by RDO from an application program.

The offline utility program, DFHCSDUP, and the EXEC CICS CSD commands are the preferred methods to examine and amend CSD files. DFHCSDUP can be used to update CSD files in bulk. DFHCSDUP can be invoked from a user program, running either in batch mode or under TSO.

To invoke the programmable interface to CEDA, use this command:

```
EXEC CICS LINK PROGRAM('DFHEDAP')
              COMMAREA(cedaparm)
```

where DFHEDAP is the name of the entry point in the RDO program, and *cedaparm* is a user-defined name of a parameter list consisting of five 31-bit addresses (each contained in a fullword) as follows:

1. Address of a field containing the RDO command in source form.
2. Address of a halfword binary field specifying the length of the command. The maximum length of the input command is 1022 bytes.
3. Address of a 1-byte indicator field defined as follows:

   X'80' Display output at terminal instead of returning it to caller.

   X'00' Do not display output at terminal.
4. Address of a field in which output is to be placed by DFHEDAP.
5. Address of a halfword binary field specifying the maximum length of output that the application can handle.

If the indicator in address 3 is X'80', output is displayed at the terminal. In this case, you can enter any number of CEDA commands at the terminal, in response to the output displayed on your screen. Control is returned to your application program when you press PF3.

However, if the indicator is X'00' (output is not to be displayed at the terminal), DFHEDAP returns control to your application program immediately after processing the RDO command specified in the first address. At the same time, DFHEDAP returns the output as one or two concatenated, structured fields. The output from a single request comprises one field for the translation stage and one or none for the execution stage. Each field has the format:

- Binary halfword containing inclusive length of field.
- Binary halfword containing the number of messages produced.
- Binary halfword containing the highest message-severity: ‘0' and ‘4' continue to execution; ‘8' and ‘12' do not continue to execution.
- Variable-length data containing:
  - For the translation stage: diagnostic messages if there are any.
  - For the execution stage: data that would normally appear on the CEDA screen, including messages. Each line begins with a new line (NL) character and, otherwise, consists of blanks and uppercase alphanumeric characters.

The format of this data is not guaranteed from release to release, but it is the same as that displayed by CEDA. (Analysis of this data should not normally be necessary. Typically, your program is interested only in whether or not the command was successful.) If the total output is longer than the maximum length specified by the user, it is truncated.

Note:

1. An attempt to start CEDA from an application program by an EXEC CICS START command must fail. This is because CEDA's first action is to request input from its associated terminal, whereas an automatically initiated transaction must first send data to the terminal.

   An attempt to start CEDA under CECI by an EXEC CICS START command fails for similar reasons.
2. The RDO command passed in address 1 of the CEDAPARM parameter list must be valid. (For example, spelling errors such as PRORGAM for PROGRAM are not corrected automatically when you use the programmable interface.)
3. You cannot use the programmable interface to change the values of CEDA keywords that are obsolete in this release of CICS, but which are retained for compatibility with earlier releases. That is, the interface does not support *compatibility mode*.
4. CEDA issues various syncpoints as part of its processing. Therefore, when your program links to DFHEDAP the current unit of work (UOW) of the transaction is completed. This may result in problems if, for example, there are outstanding browse operations against VSAM datasets.

Using DFHEDAP in a DTP environment

The LINK DFHEDAP function is intended to be used in a single environment. It is not supported in a distributed transaction programming (DTP) environment; using it in such an environment can result in abends.

In a DTP environment, CICS may attempt to propagate SYNCPOINT and SYNCPOINT ROLLBACK requests across sessions to other systems. These requests are issued by CEDA modules that are invoked by the use of LINK DFHEDAP. Issuing SYNCPOINT ROLLBACK means that LINK DFHEDAP cannot be used in a DTP environment that owns LU6.1 links.

Generally, a session should be in SEND state to initiate a SYNCPOINT, but the session may not remain in SEND state once a LINK DFHEDAP command is issued. For information about valid commands and states, see Distributed transaction processing overview in Product overview.

The code invoked by LINK DFHEDAP can result in wrong sequence of commands. For example, if the code invoked by DFHEDAP issues a SYNCPOINT ROLLBACK from a back-end application program whose session is in SEND state (and which has never issued a SYNCPOINT), the session will be put into RECEIVE state. If the code invoked by DFHEDAP then issues a SYNCPOINT, an abend occurs. This can be prevented by all DTP applications issuing a SYNCPOINT request when they get into SEND state (on all of their sessions) and before they issue the LINK DFHEDAP command.

Do not attempt to use LINK DFHEDAP when more than a pair of DTP application programs are involved—that is, one front end and one back end.

The general rules for using LINK DFHEDAP within a simple DTP environment (one front end and one back end) are that all sessions in a DTP environment should be in SEND state when the LINK DFHEDAP command is issued, and they should revert to SEND state in the event of a SYNCPOINT ROLLBACK being issued by the DFHEDAP code.

Chapter 49. User programs for the system definition utility program (DFHCSDUP)

You can write your own programs that modify or extend the CICS system definition utility program (DFHCSDUP).

An overview of DFHCSDUP

The CICS system definition utility program (DFHCSDUP) is an offline utility program that allows you to read from and write to a CICS system definition (CSD) file, either while CICS is running or while it is inactive.

Using DFHCSDUP, you can do the following:

- Add a group to the end of a named list in a CSD file
- Alter the definition of a single resource, on the CSD
- Append a group list from one CSD file to a group list in another, or in the same, CSD file
- Copy all of the resource definitions in one group to another group in the same, or in a different, CSD file
- Define a single resource, or a group of resources, on the CSD
- Delete from the CSD a single resource definition, all of the resource definitions in a group, or all of the group names in a list
- Extract requested data from the CSD and pass it to a named user program for processing
- Initialize a new CSD file, and add to it the CICS-supplied resource definitions
- List selected resource definitions, groups, and lists
- Process an APAR—that is, apply maintenance for a specific APAR to the CSD
- Remove a single group from a list on the CSD file
- Scan all IBM-supplied and user-defined groups for a resource
- Service a CSD file when necessary
- Upgrade the CICS-supplied resource definitions in a primary CSD file for a new release of CICS
- Define resources using a set of user-defined default values
- Verify a CSD file by removing internal locks on groups and lists.

You can invoke DFHCSDUP in two ways:

- As a batch program.
- From a user program running either in batch mode or in a TSO environment. "Invoking DFHCSDUP from a user program" on page 754 describes this method.

Invoking a user program from DFHCSDUP

You can invoke one of three sample programs during EXTRACT processing.

For more information about the extract command, see The DFHCSDUP EXTRACT command.

Writing a program to be invoked during EXTRACT processing

The DFHCSDUP LIST command produces reports about the current status of the CSD file that vary only according to the input parameters you provide. Another DFHCSDUP command, EXTRACT, causes the CSD data you select to be passed unformatted to a user program. The user program can then create reports of the CSD data that meet local requirements.

For example, you could cross-refer related definitions (such as TERMINALs and TYPETERMs), or you could sort the data by attribute values, such as security keys or processing priorities. The user program could also write the requested resource attributes to a data set to be used as input to a database product, such as SQL, DB2, or the Data Extract program product.

The user progrm must be linked RMODE(24). It receives control in 24-bit primary-space translation mode. (For information about translation modes, see the *IBM z/Architecture Principles of Operation* manual.) The contents of the access registers are unpredictable. The program must return control in 24-bit primary-space translation mode, and it must restore any access registers that it modifies (in addition to restoring the general purpose registers).

There are three sample programs that can be invoked from DFHCSDUP during EXTRACT processing. The sample programs, and how to replace them with your own versions, are described in "The sample EXTRACT programs" on page 746.

When the user program is invoked

The user program can be invoked at nine different points during the processing of the EXTRACT command by DFHCSDUP. However, your program is invoked at all of these points only if you specify both LIST and OBJECTS on the EXTRACT command. The invocation points are as follows:

1. At the beginning of EXTRACT processing. This is to allow for activities such as file opening and storage acquisition.
2. At the beginning of LIST processing, but only if you have specified a LIST value on the EXTRACT command.
3. At the start of every group being processed by the EXTRACT command.
4. At the start of each object (that is, resource type—TERMINAL, PROGRAM, and so on) that is being processed, to allow for selection on an object or group basis.

   Note: If you have specified LIST but not OBJECTS on the EXTRACT command, this invocation does not occur.
5. For every keyword (attribute) in the extracted object, but only if you have specified OBJECTS on the EXTRACT command. This is to allow for the detailed processing that may be necessary for cross-referencing.
6. At the end of every object—that is, when all of the keywords within an object have been processed. This is to allow for the processing of data built up from the detailed items, and it occurs once for each object.
7. At the end of every group, to allow for processing of the accumulated data.
8. At the end of LIST processing, if you have specified a LIST value on the EXTRACT command.
9. When EXTRACT processing is complete, to allow for closing of files, release of storage, and so on.

Parameters passed from DFHCSDUP to the user program

On every invocation of the user program, DFHCSDUP passes a parameter list addressed by general register 1. The parameter list consists of a series of fullwords that address the fields described in more detail here. The addresses set in the parameter list vary, depending on the point that EXTRACT processing has reached.

The parameter list contains the following fields:

Function Type Ptr
The address of a halfword field that contains a code defining the point in EXTRACT processing reached.

The function codes are as follows:

0 Initial call
2 List start call
4 Group start call
6 Object start call
8 Keyword detail call
10 Object end call
12 Group end call
14 List end call
16 Final call.

Workarea Ptr
This is the address of a field containing the address of a fullword to be used by the user application to store the address of any user-acquired work area.

Back translated command Ptr
The address of a fullword that contains the address of a 75-byte area of storage that contains the EXTRACT command that is being processed.

List name Ptr
The address of an 8-byte field that identifies the RDO list from which the current object is taken. This value is set only on the 'list start' and 'list end' calls.

Group name Ptr
The address of an 8-byte field that identifies the RDO group from which the current object is taken. This value is set on the 'group start', 'group end', 'object start', 'object end', and 'keyword' calls.

Object type Ptr
The address of a 12-byte field that identifies the type of object (such as TRANSACTION, PROGRAM, and so on), and is set only on the 'object start', 'object end', and 'keyword' calls.

Object name Ptr
The address of an 8-byte field that contains the name of the object, and is set only on the 'object start', 'object end', and 'keyword' calls.

Keyword name Ptr
The address of a 12-byte field that contains the name of the keyword being processed, and is set only on 'keyword' calls.

Keyword length Ptr
The address of a halfword field that contains the length of the value associated with the keyword, and is set only on 'keyword' calls.

Keyword Value Ptr
The address of the storage area that contains the value associated with the keyword, and is set only on 'keyword' calls.

Note: Fields not set with a pointer value contain a null value.

The sample EXTRACT programs

Three CICS-supplied sample programs can be invoked during DFHCSDUP EXTRACT processing.

Two of these are provided in COBOL, PL/I, and assembler language, and the third is provided in COBOL only.

Table 41. Sample EXTRACT user programs for the DFHCSDUP utility program

Program names	Languages	Description
DFH$CRFA DFH0CRFC DFH$CRFP	Assembler COBOL PL/I	Produces a cross-reference listing of the resource definitions defined in the group or list you specify on the EXTRACT command.
DFH$FORA DFH0FORC DFH$FORP	Assembler COBOL PL/I	Formats the group or list of resource definitions you specify on the EXTRACT command into a form suitable for the DB2 table load utility.
DFH0CBDC	COBOL	Writes the list or group of resource definitions you specify on the EXTRACT command in the form of DEFINE commands, suitable for use as a backup copy of the resources extracted.

You can use the sample programs as supplied, or as models on which to base your own programs.

The assembler-language and COBOL versions of the CSD cross-referencing program, DFH$CRFA and DFH0CRFC respectively, are supplied in executable form in CICSTS52.CICS.SDFHLOAD. The PL/I version, DFH$CRFP, is supplied in source form only.

The assembler-language and COBOL versions of the DB2 formatting program, DFH$FORA and DFH0FORC respectively, are supplied in executable form in CICSTS52.CICS.SDFHLOAD. The PL/I version, DFH$FORP, is supplied in source form only.

The CSD backup utility program, DFH0CBDC, is supplied in executable form in CICSTS52.CICS.SDFHLOAD.

The source statements of all versions of all the sample programs are supplied in CICSTS52.CICS.SDFHSAMP.

The CICS-supplied sample DB2 formatting programs (DFH$FORx) cannot be used when the CSD compatibility option (COMPAT) is specified on the DFHCSDUP utility program. The output from the CSD cross-reference listing and CSD backup utility programs depends on whether the compatibility option is specified. If the compatibility option is specified, the output includes obsolete keywords from releases before CICS Transaction Server for z/OS, Version 5 Release 2; if the option is not specified, only keywords from CICS Transaction Server for z/OS, Version 5 Release 2 are output.

All of the sample extract user programs support the definition signature fields.

Note that the sample programs require you to specify the OBJECTS keyword on the DFHCSDUP EXTRACT command.

The output data definition names (ddnames) for the sample programs are as follows:

CRFOUT
CSD cross-referencing program

FOROUT
DB2 formatting program

CBDOUT
CSD backup utility program.

The sample programs are discussed in the next three sections.

The CSD cross-referencing program

Use the CICS-supplied sample CSD cross-referencing program to produces a cross-reference listing of the resource definitions defined in the group or list specified on the EXTRACT command. Run the DFH$CRFA version of the cross-referencing program if you are using assembler, the DFH0CRFC version if you are using COBOL, and the DFH$CRFP version if you are using PL/I.

The CICS-supplied sample CSD cross-referencing program produces a cross-reference listing of objects and keywords on the CSD. The data gathered by the EXTRACT command is passed to the sample program, where it is saved in a cross-reference table. On the final call to this sample program, the contents of the table are printed in collating sequence.

The program must be run against an EXTRACT command of the form:

```
EXTRACT GROUP(group name) OBJECTS USERPROGRAM(program-name)
```

or:

```
EXTRACT LIST(list name) OBJECTS USERPROGRAM(program-name)
```

Note that the sample program requires you to specify the OBJECTS keyword.

For this program only, in addition to the EXTRACT command, you must define, in a sequential data set, the objects and keywords for which you want a cross-reference listing. The data set is read by the sample program using the ddname CRFINPT.

CRFINPT is a sequential file containing 80-byte records. Each record contains one object or keyword to be cross-referenced. You can cross-reference any valid resource type or attribute known to CEDA. For example, your CRFINPT file may contain the following entries (one per line):

PROGRAM
TRANSACTION
TYPETERM
DSNAME

Note that a maximum of ten entries can be included in the CRFINPT file when using the COBOL sample program, DFH0CRFC.

For each record in the file, a report is produced detailing the different values assigned to the keyword, where they are defined, and where they are used. Note that keyword values longer than 44 characters are truncated.

You should define the DCB subparameters for CRFINPT as DSORG=PS, RECFM=F, LRECL=80, and BLKSIZE=80.

The DB2 formatting program

Use the CICS supplied sample DB2 formatting program to organize CSD data into a format suitable for the DB2 table load utility. Run the DFH$FORA sample program if you are using assembler, the DFH0FORC sample program if you are using COBOL, and the DFH$FORP sample program if you are using PL/I.

The CICS-supplied sample DB2 formatting program organizes the data into columns that correspond to the columns defined in the load utility's input. Each selected resource causes a record to be written to this program's output file, with the first 4 characters identifying the resource type.

The program must be run against an EXTRACT command of the form:

```
EXTRACT GROUP(group name) OBJECTS USERPROGRAM(program-name)
```

or:

```
EXTRACT LIST(list name) OBJECTS USERPROGRAM(program-name)
```

Note that the sample program requires you to specify the OBJECTS keyword.

Storing CSD data in DB2:

If you want to store data in a DB2 database, you must format the data, create tables in DB2, and populate those tables with the data that has been formatted.

About this task

If you want to store the CSD data output by DFHCSDUP in DB2, complete the following steps:

Procedure

1. Create tables in DB2. Use the sample DB2 table definitions provided in SDFHSAMP(DFH$DB2T) to create tables in DB2.
   a. Update the DFH$DB2T sample. Update the DFH$DB2T sample code to replace occurrences of <*data base name*>.<*table space*> with a database and table space name relevant to your environment, for example TESTDB.TESTDBTSP.
2. Run DFHCSDUP with the DB2 formatting program. Use the supplied sample DB2 formatting program to organize the CSD data from DFHCSDUP into a format suitable for the DB2 table load utility. There are three versions of the sample formatting program; DFH$FORA for assembler, DFH0FORC for COBOL, and DFH$FORP for PL/I. The FOROUT DD statement must point to a data set with a record size of 1536.
3. Populate the DB2 tables with the formatted data. Use the sample DB2 load logic provided in SDFHSAMP(DFH$SQLT) to populate the tables in DB2. A DD card called SYSREC is required, which must point to the data set that the DB2 formatting program has written its output to. This is the same data set that the FOROUT DD pointed to in the previous step.

a. Update the DFH$SQLT sample. Update the DFH$SQLT sample code to replace occurrences of *<owner>* with an authorized user ID for the database.

Results

The data is stored in the tables you created in DB2.

Example

Here is an example of the JCL code that you could use to format your data and move it to DB2:

```
//DB2EXT  JOB MSGCLASS=H,CLASS=A,NOTIFY=&SYSUID
//*-------------------------------------------
//*CSD EXTRACT
//*-------------------------------------------
//EXTRACT  EXEC PGM=DFHCSDUP,REGION=2000K
//SYSPRINT DD SYSOUT=A
//*
//STEPLIB  DD DSN=<cicshlq>.SDFHLOAD,DISP=SHR
//DFHCSD   DD DSN=TEST.RTSREG.CTS410.CICS660.CSD,DISP=SHR
//SYSOUT   DD *
//FOROUT   DD DSN=USER1.TEST.DB2.INPUT,DISP=SHR
//SYSIN    DD *
  EXTRACT GROUP(TESTGRP) USERPROGRAM(DFH0FORC) OBJECTS
/*
//
//DB2STG JOB 1,USER=TEST,CLASS=A,MSGCLASS=A,
//          MSGLEVEL=(1,1),REGION=7M,NOTIFY=&SYSUID
//*
//JOBLIB   DD DSN=SYS2.DB2.V910.SDSNLOAD,DISP=SHR
//*-------------------------------------------
//*   CREATE STORAGE GROUP/DATABASES/TABLESPACES
//*-------------------------------------------
//CREATDB  EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  DSN SYSTEM(DHP2)
  RUN PROGRAM(DSNTIAD) PLAN(DSNTIA91) -
      LIB('DSN910P2.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
  DROP     DATABASE TESTDB;
  DROP     STOGROUP TESTDBST;
  CREATE   STOGROUP TESTDBST VOLUMES (SYSDA) VCAT DSN910P2;
  CREATE   DATABASE TESTDB STOGROUP TESTDBST;
  COMMIT;
  CREATE   TABLESPACE TESTDBTS IN TESTDB
  LOCKSIZE ROW;
  COMMIT;
/*
//
//*-------------------------------------------
//*  CREATE TABLES AND INDEXES
//*-------------------------------------------
//CREATTAB EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB  DD DSN=DSN910P2.DBRMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  DSN SYSTEM(DHP2)
  RUN PROGRAM(DSNTIAD) PLAN(DSNTIA91) -
  LIB('DSN910P2.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
```

```
//* INCLUDE CONTENTS OF SDFHSAMP(DFH$DB2T) HERE
//*e.g.
//*CREATE TABLE ATOMSERVICE
//*   (ATOMSERVICE   CHAR(8),
//*    RDOGROUP      CHAR(8),
//*    DESCRIPTION   CHAR(58),
//*    ATOMTYPE      CHAR(10),
//*    STATUS        CHAR(8),
//*    CONFIGFILE    CHAR(255)
//*    RESOURCENAME  CHAR(16),
//*    RESOURCETYPE  CHAR(7),
//*    BINDFILE      CHAR(255)
//*    DEFINETIME    CHAR(17),
//*    CHANGETIME    CHAR(17),
//*    CHANGEUSRID   CHAR(8),
//*    CHANGEAGENT   CHAR(8),
//*    CHANGEAGREL   CHAR(4))
//*   IN TESTDB.TESTDBTSP;
//*
//*CREATE INDEX ATOMI ON ATOMSERVICE
//*   (ATOMSERVICE ASC);
//*
//* COMMIT;
//* etc ...
//
//GRANTACC EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  DSN SYSTEM(DHP2)
  RUN PROGRAM(DSNTIAD) PLAN(DSNTIA91) -
     LIB('DSN910P2.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
  GRANT DBADM ON DATABASE TESTDB TO PUBLIC;
  GRANT USE OF TABLESPACE TESTDB.TESTDBTS TO PUBLIC;
  GRANT ALL PRIVILEGES ON TABLE TESTDBTB TO PUBLIC;
/*
//
//*---------------------------------------------
//*LOAD DATA INTO TABLES
//*---------------------------------------------
//DB2TBL JOB  CLASS=A,MSGCLASS=H,NOTIFY=&SYSUID,REGION=4096K,
//          USER=TEST
//*
//JOBLIB DD DSN=SYS2.DB2.V910.SDSNLOAD,DISP=SHR
//*
//*
//****************  LOAD ITMNUMBR - TRANS WKLD  ********************
//*
//STEPITM     EXEC  PGM=DSNUTILB,PARM='DHP2'
//UTPRINT  DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1   DD DSN=ST.ITM02.SYSUT1,DISP=(MOD,DELETE,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(10,5))
//SYSREC   DD DSN=USER1.TEST.DB2.INPUT,DISP=SHR
//SORTOUT  DD UNIT=SYSDA,SPACE=(CYL,(40,10),,,ROUND)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SORTWK04 DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSIN DD *    << INCLUDE CONTENTS OF SDFHSAMP(DFH$SQLT) HERE >>
//* e.g.
//*LOAD DATA
//*RESUME NO REPLACE
//*INTO TABLE ATOMSERVICE
```

```
//*WHEN (1:4) = 'ATOM'
//*(ATOMSERVICE    POSITION (5:12)          CHAR,
//* RDOGROUP       POSITION (13:20)         CHAR,
//* DESCRIPTION    POSITION (21:78)         CHAR,
//* ATOMTYPE       POSITION (79:88)         CHAR,
//* STATUS         POSITION (89:96)         CHAR,
//* CONFIGFILE     POSITION (97:351)        CHAR,
//* RESOURCENAME   POSITION (352:367)       CHAR,
//* RESOURCETYPE   POSITION (368:374)       CHAR,
//* BINDFILE       POSITION (375:629)       CHAR,
//* DEFINETIME     POSITION (630:646)       CHAR,
//* CHANGETIME     POSITION (647:663)       CHAR,
//* CHANGEUSRID    POSITION (664:671)       CHAR,
//* CHANGEAGENT    POSITION (672:679)       CHAR,
//* CHANGEAGREL    POSITION (680:683)       CHAR)
//*INTO TABLE BUNDLE
//*
//* etc...
/*
//
```

The CSD backup utility program

Use the CICS-supplied sample CSD backup utility program, DFH0CBDC, to write the list or group of resource definitions specified on the EXTRACT command in the form of DEFINE commands that are suitable for use as a backup copy.

The CICS-supplied sample CSD backup utility program produces a file of DFHCSDUP DEFINE control statements. The file can be used:

- For later editing and commenting to document CSD resources
- For distribution, in part or as a whole, to other CICS installations
- To re-create or add resource definitions to any CSD using DFHCSDUP.

The program must be run against an EXTRACT command of the form:

```
EXTRACT GROUP(group name) OBJECTS USERPROGRAM(program-name)
```

or:

```
EXTRACT LIST(list name) OBJECTS USERPROGRAM(program-name)
```

Note that the sample program requires you to specify the OBJECTS keyword.

Note the following points when using DFH0CBDC:

- It can deal with only one set of data during each invocation of DFHCSDUP; if two EXTRACT commands are issued, the second set of data overwrites the first.
- In the file produced by DFH0CBDC, any DEFINE statements that relate to CICS-supplied resources are preceded by an asterisk (*) in column 1; in other words, they are commented out. This is important if you use the file as input to define resources to a CSD. (The CICS-supplied definitions are already present in the CSD, having been produced automatically when it was initialized.)
- If you remove an asterisk from column 1 (to reinstate the DEFINE statement), do so by deleting it, **not** by overtyping it with a blank. This ensures that the resulting command is no more than 72 characters long; if it is longer than this, errors occur when the output is passed back through DFHCSDUP.

Assembling and link-editing EXTRACT programs

You must assemble (or compile) and link-edit DFHCSDUP user programs as batch programs, not as CICS applications, and you need link-edit control statements appropriate to the language in which they are written.

About this task

Note: DFHCSDUP user programs should not be translated, or unpredictable results could occur.

When you compile the COBOL versions of the sample programs, you must specify the compiler attributes RENT and NORES. [3]

When you link-edit the programs, you must specify the following link-edit control statements:

- An ENTRY statement that defines the entry name as DFHEXTRA
- An INCLUDE statement for a CICS-supplied stub that must be included in your user program
- A CHANGE statement to change the dummy CSECT name in the CICS-supplied stub from EXITEP to the name of your user program.

When you link-edit the COBOL versions of the sample programs, you must specify RMODE(24).

These requirements are explained in more detail for each of the languages (assembler, COBOL, and PL/I) shown in the following sample job streams.

An assembler language version

The sample job in Figure 115 shows the link-edit statements you need for an assembler language version of a DFHCSDUP user program.

```
//DFHCRFA  JOB (accounting information),CLASS=A,MSGCLASS=A,NOTIFY=userid
//* .
//* Assembler job step here
//*  .
//LINK EXEC PGM=IEWL,PARM='XREF,LIST,LET'
//OBJLIB   DD DSN=object.module.library,DISP=SHR
//SYSLIB   DD DSN=CICSTS52.CICS.SDFHLOAD,DISP=SHR
//SYSLMOD  DD DSN=user.library,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSPRINT DD SYSOUT=A
//SYSLIN   DD *
 ENTRY     DFHEXTRA                                          1
 CHANGE    EXITEP(csectname)                                 2
 INCLUDE   SYSLIB(DFHEXAI)                                   3
 INCLUDE   OBJLIB(obj-name)                                  4
 NAME      progname(R)                                       5
```

Figure 115. Link-edit control statements for a DFHCSDUP user program (assembler language)

Notes for the assembler job:

1 Specify the entry name as DFHEXTRA, which is the entry name in the CICS-supplied stub, DFHEXAI. (See 3.)

2 The CICS-supplied stub, DFHEXAI, is generated with a link to the user program using a dummy CSECT name (EXITEP). Use the link-edit CHANGE statement to change the CSECT name from EXITEP to the name of the CSECT in the user program. In the two CICS-supplied assembler language sample programs, these names are:

3. The RENT compiler attribute prevents an abend C03 ('Data set was not closed properly') occurring after the sample program receives an abend such as B37 ('Data set size is smaller than output').

CREFCSD
The CSECT name in DFH$CRFA, the cross-reference listing user program.

FORMCSD
The CSECT name in DFH$FORA, the DB2-formatting user program.

3 Include DFHEXAI in any assembler language user program that you write for use with the DFHCSDUP EXTRACT command. DFHEXAI is the interface stub between DFHCULIS, a module in DFHCSDUP, and the user program.

4 `obj-name` is the name of the library member that contains the assembled object module.

5 `progname` is the name you want to call the load module; this is the name that you specify on the USERPROGRAM parameter of the EXTRACT command.

A Language Environment version

The sample job in Figure 116 shows the link-edit statements you need for a DFHCSDUP user program written in a Language Environment-conforming high-level language.

```
//DFHCRFA  JOB (accounting information),CLASS=A,MSGCLASS=A,NOTIFY=userid
//*  .
//* Compile job step here
//*  .
//LINK EXEC PGM=IEWL,PARM='XREF,LIST,LET'
//SYSLIB   DD DSN=PP.ADLE370.OS39025.SCEELKED
//CICSLIB  DD DSN=CICSTS52.CICS.CICS.SDFHLOAD,DISP=SHR
//OBJLIB   DD DSN=object.module.library,DISP=SHR
//SYSLMOD  DD DSN=user.library,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSPRINT DD SYSOUT=A
//SYSLIN   DD *
 ENTRY     DFHEXTRA                                          1
 CHANGE    EXITEP(prof-id)                                   2
 INCLUDE   CICSLIB(DFHEXLE)                                  3
 INCLUDE   OBJLIB(obj-prog)                                  4
 NAME      progname(R)                                       5
```

Figure 116. Link-edit control statements for a DFHCSDUP user program (Language Environment)

Notes for the Language Environment job:

1 Specify the entry name as DFHEXTRA, which is the entry name in the CICS-supplied stub, DFHEXLE (see 3).

2 The CICS-supplied stub, DFHEXLE, is generated with a link to the user program using a dummy CSECT name (EXITEP). Use the link-edit CHANGE statement to change the CSECT name from EXITEP to the name specified on the PROC statement in the user program.

3 Include DFHEXLE in any LE-conforming user program that you write for use with the DFHCSDUP EXTRACT command. DFHEXLE is the interface stub between DFHCULIS, a module in DFHCSDUP, and the Language Environment user program.

4 `obj-prog` is the name of the object program.

5 `progname` is the name you want for the load module; this is the name that you specify on the USERPROGRAM parameter of the EXTRACT command.

Invoking DFHCSDUP from a user program

You can invoke DFHCSDUP from a user program, enabling you to create a flexible interface to the utility.

By specifying the appropriate entry parameters, your program can cause DFHCSDUP to pass control to an exit routine at any of five exit points. The exits can be used, for example, to pass commands to DFHCSDUP, or to respond to messages produced by DFHCSDUP processing.

You can run your user program:

- In batch mode
- Under TSO.

Note:

1. In a TSO environment, it is normally possible for the terminal operator to interrupt processing at any time by means of an ATTENTION interrupt. In order to protect the integrity of the CSD file, DFHCSDUP does not respond to such an interrupt until after it has completed the processing associated with the current command. It then writes message number 'DFH5618' to the put-message exit (see "The put-message exit" on page 760), where this is available, and also to the default output file:

   ```
   AN ATTENTION INTERRUPT WAS
   REQUESTED DURING DFHCSDUP PROCESSING
   ```

   Your put-message exit routine can terminate DFHCSDUP, if desired. (Note that you **must** supply a put-message routine if you want your operators to regain control after an ATTENTION interrupt.)
2. Suitably authorized TSO operators can use the CEDA INSTALL transaction to install resources that have previously been defined with DFHCSDUP.

Entry parameters for DFHCSDUP

When your program invokes DFHCSDUP, it passes a parameter list addressed by register 1. The program can pass up to five parameters.

OPTIONS
A list of character strings, separated by commas. This information is the information that would otherwise be passed on the PARM keyword of the EXEC statement of JCL. A maximum of four options can be specified:

CSD({READWRITE | READONLY})
Specifies whether you require read-write or read-only access to the CSD.

PAGESIZE(*nnnn*)
Specifies the number of lines per page on output listings. Valid values are 4 through 9999. The default value is 60.

NOCOMPAT | COMPAT
Specifies whether DFHCSDUP is invoked in compatibility mode. By default, it is not invoked in compatibility mode. For details of compatibility mode, see Sharing the CSD between different releases of CICS in Configuring.

UPPERCASE
Specifies that output listings are printed entirely in uppercase characters. The default is to print in mixed case.

DDNAMES
A list of data definition names (ddnames) that, if specified, are substituted for those normally used by DFHCSDUP.

HDING
The starting page number of any listing produced by DFHCSDUP. You can use this parameter to ensure that subsequent invocations produce logically numbered listings. If this parameter is not specified, the starting page number is set to 1.

The length of the page number data (field `bb` in Figure 117 on page 756) must be 0 or 4. The page number, if supplied, must be four numeric EBCDIC characters. The field, if present, is updated upon exit from DFHCSDUP with a number one greater than that of the last page printed.

DCBS
The addresses of a set of data control blocks for use internally by DFHCSDUP. Any DCBs (or ACBs) that you specify are used internally, instead of those normally used by DFHCSDUP.

Note that if you specify both replacement ddnames and replacement DCBs, the alternative DCBs are used, but the alternative ddnames are disregarded.

EXITS
The addresses of a set of user exit routines to be invoked during processing of DFHCSDUP.

The structure of the parameter list is shown in Figure 117 on page 756.

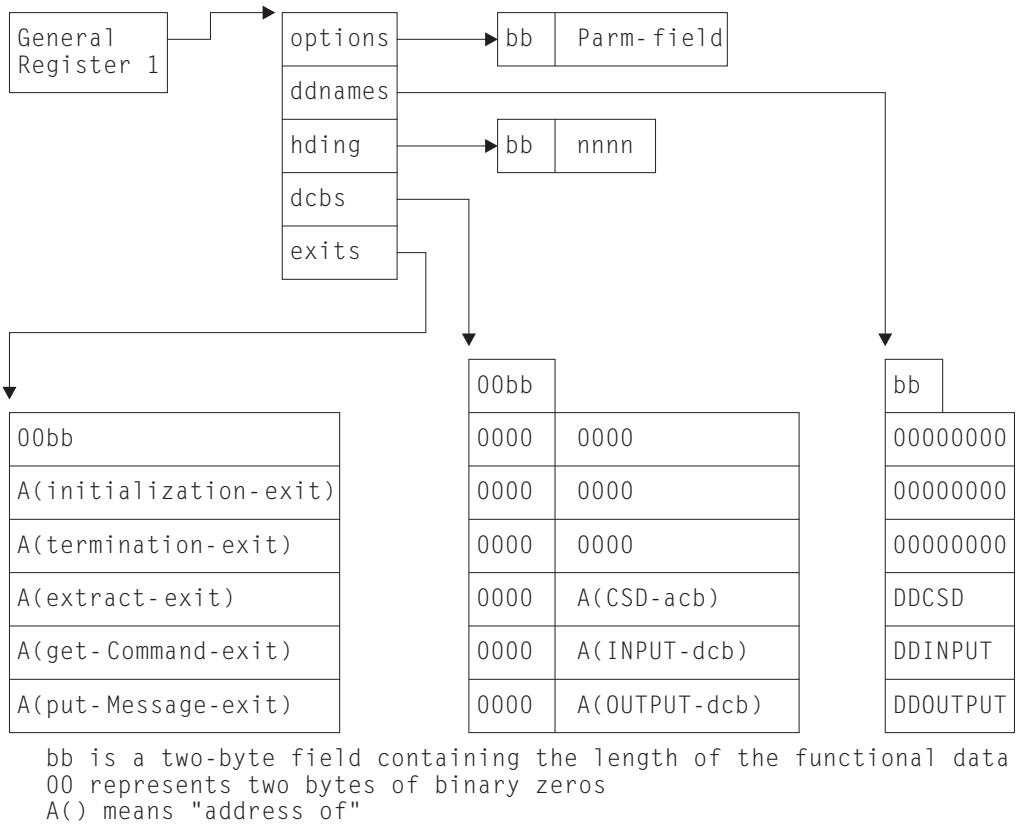


Figure 117. Entry parameters for DFHCSDUP

You should note the following:

- Each parameter contains a length field, followed by some functional data.
- The functional data for the DDNAMES, DCBS, and EXITS parameters contains multiple subentries.
- The parameters OPTIONS, DDNAMES, and HDING are aligned on a halfword boundary, and the first two bytes `bb` contain the binary number of **bytes** in the following functional data.
- The parameters DCBS and EXITS are aligned on a fullword boundary, and the first four bytes '00bb' contain the binary number of **fullwords** in the following functional data.
- If the `bb` field for any parameter is zero, the parameter is ignored.
- If a subentry in the functional data is all binary zeros, it is ignored.
- If any subentry is not within the length indicated by `bb`, it is ignored.
- In the DDNAMES functional data, each subentry consists of an 8-byte ddname to replace a default ddname used by DFHCSDUP. DFHCSDUP does not use the first three subentries of the DDNAMES parameter. The fourth, fifth, and sixth subentries, if present, replace the ddnames of DFHCSD, SYSIN, and SYSPRINT, respectively.
- In the DCBS functional data, each subentry consists of two fullwords. The first word is not used by CICS. The second word contains the address of an open DCB or ACB. You must ensure that the DCB or ACB has been opened with the correct attributes, which are as follows:

  PRIMARY CSD
  AM=VSAM,MACRF=(KEY,DIR,SEQ,IN,OUT)

INPUT FILE
DSORG=PS,MACRF=GL,LRECL=80,RECFM=FB

The address of any end-of-data routine (EODAD) or I/O error routine (SYNAD) in the DCB is overlaid by DFHCSDUP.

OUTPUT FILE
DSORG=PS,MACRF=PL,LRECL=125,RECFM=VBA

DFHCSDUP does not use the first three subentries of the DCBS parameter. The fourth, fifth, and sixth subentries, if present, are used instead of the internal DCBs or ACBs for DFHCSD, SYSIN, and SYSPRINT, respectively.

- In the EXITS parameter, each subentry consists of a single fullword containing the address of an exit routine. You must specify the exit routines in the order shown in Figure 117 on page 756. (The user exits are described in "The user exit points in DFHCSDUP.")

Responsibilities of the user program

Before invoking DFHCSDUP, your calling program must ensure that:

- RMODE(24) is in force.
- Operating system register conventions are obeyed.
- If the EXITS parameter is passed, any programming environment needed by the exit routines has been initialized.
- Any ACBs or DCBs passed for use by DFHCSDUP are OPEN.

The user exit points in DFHCSDUP

There are five user exit points in DFHCSDUP. By specifying the appropriate entry parameters, you can cause DFHCSDUP to pass control to an exit routine at any of these points.

None of the user exits supports XPI calls.

Parameters passed to the user exit routines

The address of a parameter list is passed to the user exit routine in register 1. The list contains some standard parameters that are passed to all of the exit routines, and may also contain some exit-specific parameters that are unique to the exit point from which the exit routine is being invoked.

The standard parameter list is different from that used by CICS global user exits. The following DFHUEXIT DSECT maps the standard parameter list used by DFHCSDUP and the sample program DFH$CUS1. (The UEPCMDA and UEPCMDL fields are are used only by the get-command exit.)

```
DFHUEXIT DSECT
UEPEXN   DS    A               ADDRESS OF EXIT NUMBER
UEPGAA   DS    A               ADDRESS OF GLOBAL AREA
UEPGAL   DS    A               ADDRESS OF GLOBAL AREA LENGTH
UEPCRCA  DS    A               ADDRESS OF CURRENT RETURN-CODE
UEPTCA   DS    A               ADDRESS OF TCA
UEPCSA   DS    A               ADDRESS OF CSA
UEPHMSA  DS    A               ADDRESS OF SAVE AREA USED BY HOST
UEPSTACK DS    A               ADDRESS OF KERNEL STACK ENTRY
UEPXSTOR DS    A               ADDRESS OF STORAGE OF XPI PARMS
UEPTRACE DS    A               ADDRESS OF TRACE FLAG
*
UEPCMDA  DS    A               ADDRESS OF UTILITY COMMAND
UEPCMDL  DS    A               ADDRESS OF LENGTH OF UTILITY
*                              COMMAND
```

Explanations of the exit-specific parameters are included in the descriptions of the individual exits.

The initialization exit

The initialization exit is invoked once during DFHCSDUP initialization. Its purpose is to allow a routine to perform exit-related initialization.

For example, the routine may obtain its own global work area and save its address in UEPGAA and its length in the halfword pointed to by UEPGAL. These values are retained by DFHCSDUP and become available at the other exit points.

When invoked
Invoked once, on entry to DFHCSDUP.

Exit-specific parameters
None.

Return codes

UERCNORM (X'00')
Continue processing.

UERCERR
Irrecoverable error. This causes DFHCSDUP to terminate with a return code of '8'.

XPI calls
Must not be used.

The get-command exit

The purpose of the get-command exit is to read in command lines. If it is specified, no commands are read from SYSIN.

On invocation, your exit routine must supply the address and length of a **complete** command. It must return control with either the normal return code 'UERCNORM' or with the code 'UERCDONE', signifying that it has no more commands to pass. After it has processed each command, DFHCSDUP reinvokes the exit until return code 'UERCDONE' is received.

When invoked
Invoked multiple times, at the point where DFHCSDUP would otherwise read commands from SYSIN.

Exit-specific parameters

UEPCMDA
Pointer to the address of a command.

UEPCMDL
Address of a halfword containing the length of the command text. The maximum length that can be specified is 1536 bytes.

Return codes

UERCNORM (X'00')
Continue processing.

UERCDONE (X'04')
No more commands to process. (This is equivalent to reaching end-of-file on the SYSIN file.)

UERCERR
Irrecoverable error. This causes DFHCSDUP to terminate with a return code of '8'.

XPI calls
Must not be used.

The extract exit

The extract exit is invoked at various points during processing of the EXTRACT command.

The points are listed in "When the user program is invoked" on page 744.

Note:

1. If you do not specify an EXTRACT user exit routine on the entry linkage to DFHCSDUP, or on the USERPROGRAM keyword, a syntax error occurs.
2. A user exit routine specified on the USERPROGRAM keyword is used in preference to one specified on the entry linkage.

When invoked
Invoked multiple times during processing of the EXTRACT command.

Exit-specific parameters

EXTRACT_FUNCTION_CODE_PTR
Address of a halfword containing a code that defines the point in EXTRACT processing reached. The EXTRACT function codes are listed in EXTRACT function codes.

EXTRACT_WORK_AREA_PTR
Address of a fullword containing the address of the EXTRACT work area.

EXTRACT_BACKTRAN_COMMAND_PTR
Address of a fullword containing the address of the EXTRACT command being processed.

EXTRACT_CSD_LIST_NAME_PTR
Address of an 8-byte field containing the name of the list whose data is being extracted. This value is set only on 'list start' and 'list end' calls.

EXTRACT_CSD_GROUP_NAME_PTR
Address of an 8-byte field containing the name of the group whose data is being extracted. This value is set on 'group start', 'group end', 'object start', 'object end', and 'keyword' calls.

EXTRACT_CSD_OBJECT_TYPE_PTR
Address of a 12-byte field that identifies the type of object (such as TRANSACTION, PROGRAM, and so on). This value is set only on 'object start', 'object end', and 'keyword' calls.

EXTRACT_CSD_OBJECT_NAME_PTR
Address of an 8-byte field containing the name of the object. This value is set only on 'object start', 'object end', and 'keyword' calls.

EXTRACT_KEYWORD_NAME_PTR
Address of an 12-byte field containing the name of the keyword being processed. This value is set on 'keyword' calls only.

EXTRACT_KEYWORD_LENGTH_PTR
Address of a halfword containing the length of the value associated with the keyword. This value is set on 'keyword' calls only.

EXTRACT_KEYWORD_VALUE_PTR
Address of a character string which contains the value associated with the keyword. This value is set on 'keyword' calls only.

Note that these parameters are similar to those passed when DFHCSDUP is invoked as a batch program. (See "Parameters passed from DFHCSDUP to the user program" on page 745.) However, when DFHCSDUP is invoked from a user program, the parameter list also includes the standard parameters mentioned under "Parameters passed to the user exit routines" on page 757.

Return codes

UERCNORM (X'00')
Continue processing.

UERCERR
Irrecoverable error. This causes DFHCSDUP to terminate with a return code of '8'.

XPI calls
Must not be used.

The put-message exit

The put-message exit is invoked whenever DFHCSDUP issues a message.

If you are running under TSO, you could use this exit to terminate DFHCSDUP after the operator inputs an ATTENTION interrupt. (See "Invoking DFHCSDUP from a user program" on page 754.) Or you could use it to provide messages in the operator's national language.

Even if this exit is supplied, messages are always additionally written to the default output file (that is, to SYSPRINT, or to the replacement ddname specified on the entry linkage to DFHCSDUP).

When invoked
Invoked when a message is to be issued.

Exit-specific parameters

UEPMNUM
Address of a 4-character field containing the message number

UEPMDOM
Reserved

UEPINSN
Address of a 2-byte field containing the number of insert fields

UEPINSA
Address of the following message structure:

```
               DS  F    Reserved
INS_1_TEXT_PTR  DS  A    Address of insert 1
INS_1_LEN_PTR   DS  A    Address of a fullword containing
                         the length of insert 1
                DS  F    Reserved
                DS  F    Reserved
```

```
INS_2_TEXT_PTR  DS  A    Address of insert 2
INS_2_LEN_PTR   DS  A    Address of a fullword containing
                         the length of insert 2
                DS  F    Reserved
                ...
                DS  F    Reserved
INS_n_TEXT_PTR  DS  A    Address of insert n
INS_n_LEN_PTR   DS  A    Address of a fullword containing
                         the length of insert n
                DS  F    Reserved
```

The exit-specific parameters provide a message number and insert fields only, to enable you to provide messages in the language of your TSO operators. The structure pointed to by UEPINSA is repeated as many times as UEPINSN requires.

Return codes

UERCNORM (X'00')
Continue processing.

UERCERR
Irrecoverable error. This causes DFHCSDUP to terminate with a return code of '8'.

XPI calls
Must not be used.

The termination exit

The purpose of the termination exit is to allow you to perform final housekeeping duties. It is invoked before a normal or an abnormal termination of DFHCSDUP.

When invoked
Invoked once, before termination of DFHCSDUP.

Exit-specific parameters

UEPTRMFL
Address of a 1-byte field that indicates the mode of termination. Its possible values are:

X'00' Normal termination

X'F0' Abnormal termination.

Your exit program cannot reset the value in this field.

Return codes

UERCNORM (X'00')
Continue processing.

UERCERR
Irrecoverable error. This causes DFHCSDUP to terminate with a return code of '8'.

XPI calls
Must not be used.

The sample program, DFH$CUS1

The CICS-supplied sample program, DFH$CUS1, illustrates how DFHCSDUP can be invoked from a user program. It is written as a command processor (CP) for execution under the TSO/E operating system.

Note that DFH$CUS1 uses different DCB and ACB names from those normally used by DFHCSDUP. Ensure that these are allocated before running the program under TSO/E.

Although DFH$CUS1 is intended to be run from TSO, you can also run it from, for example, a REXX EXEC. Before doing so, ensure that the load library that contains DFH$CUS1, DFHCSDUP, and DFHEITCU is in the user's search chain, LOGON proc, or linklist. Figure 118 is an example REXX EXEC that invokes DFH$CUS1.

```
/*REXX*/
"ALLOCATE DSN('XXXXX.CICS690.DFHCSD') DD(ALTACB) SHR"
"ALLOC DD(SIN) DA(*) BLKSIZE(80)"
"ALLOC DD(SPRINT) DA(*) BLKSIZE(80)"
X = PROMPT('ON')
ADDRESS TSO "DFH$CUS1"
"FREE DD(ALTACB)"
"FREE DD(SIN)"
"FREE DD(SPRINT)"
RETURN 0
```

Figure 118. A REXX program that invokes the DFH$CUS1 sample program

Part 11. Appendixes

Appendix A. XPI functions (by domain)

The XPI functions are grouped according to their functional relationships. Grouping is generally based on the CICS domain in which the XPI functions occur.

To understand the syntax that is used to describe the XPI functions, see "XPI syntax" on page 393.

Business application manager domain XPI function

The XPI provides one business application manager domain function. This is the DFHBABRX call INQUIRE_ACTIVATION.

The INQUIRE_ACTIVATION call

The INQUIRE_ACTIVATION function is provided on the DFHBABRX macro call. Use the INQUIRE_ACTIVATION call to obtain the activity name and the process type for the business transaction activity of the current transaction.

INQUIRE_ACTIVATION

```
DFHBABRX [CALL,]
    [CLEAR,]
    [IN,
    FUNCTION(INQUIRE_ACTIVATION),
    [TRANSACTION_TOKEN(name8),]]
    [RETURNED_ACTIVITYID(buffer_descriptor)]
    [RETURNED_PROCESS_NAME(buffer_descriptor)]
    [OUT,
    [ACTIVITY_NAME(name16)]
    [PROCESS_TYPE(name8)]
    RESPONSE (name1 | *),
    REASON (name1 | *)]
```

This command is threadsafe.

ACTIVITY_NAME(name16)
Returns the 16-character, user-assigned, name of the BTS activity.

PROCESS_TYPE(name8)
Returns the 8-character identifier of the type definition of the BTS process.

RETURNED_ACTIVITYID(buffer_descriptor)
Returns the 52-character, CICS-assigned, identifier of the BTS activity. RETURNED_ACTIVITYID is an output parameter (BAM returns it) and the data type is buffer, so the caller must supply an area to be used as a buffer as input to the call.

RETURNED_PROCESS_NAME(buffer_descriptor)
Returns the 36-character name of the BTS process. RETURNED_PROCESS_NAME is an output parameter (BAM returns it) and the data type is buffer, so the caller must supply an area to be used as a buffer as input to the call.

TRANSACTION_TOKEN(name8)
Specifies the transaction token for the task being inquired on.

RESPONSE and REASON values for INQUIRE_ACTIVATION

RESPONSE	*REASON*
OK	None
EXCEPTION	ACTIVITY_NOT_FOUND
DISASTER	None
INVALID	INVALID_BUFFER_LENGTH
KERNERROR	None
PURGED	None

For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

Directory domain XPI functions

The XPI provides directory domain functions that you can use to open and close an LDAP session, browse results for credentials, scan and locate results, close the browse, return the correct value and close the search.

The directory domain functions are the following DFHDDAPX calls:
- BIND_LDAP
- END_BROWSE_RESULTS
- FLUSH_LDAP_CACHE
- FREE_SEARCH_RESULTS
- GET_ATTRIBUTE_VALUE
- GET_NEXT_ATTRIBUTE
- GET_NEXT_ENTRY
- SEARCH_LDAP
- START_BROWSE_RESULTS
- UNBIND_LDAP

For more information about the use of these functions, see the *CICS Internet Guide*.

The BIND_LDAP call

The BIND_LDAP call establishes a session with an LDAP server.

The LDAP server is identified by one of the following:
- The LDAP URL and the distinguished name and password of the user authorized to extract the expected data.
- A RACF profile in the LDAPBIND class that contains the LDAP URL and distinguished name and password. This is the preferred option, as you do not need to code LDAP credentials in your application.

BIND_LDAP

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(BIND_LDAP),
    {LDAP_BIND_PROFILE(block-descriptor)|
    LDAP_SERVER_URL((block-descriptor),DISTINGUISHED_NAME((block-descriptor),
    PASSWORD(block-descriptor),}
    [CACHE_SIZE(name4),CACHE_TIME_LIMIT(name4),]]
    [OUT,
```

```
LDAP_SESSION_TOKEN(name4),
[LDAP_RESPONSE(name4),]
RESPONSE(name1 | *),
REASON(name1 | *)]
```

This command is threadsafe.

CACHE_SIZE(name4)
a fullword that specifies the number of bytes available for caching LDAP search results. A value of zero indicates an unlimited cache size. If CACHE_SIZE is specified, CACHE_TIME_LIMIT must also be specified. If neither parameter is specified, results will not be cached.

CACHE_TIME_LIMIT(name4)
a fullword that specifies the amount of time (in seconds) that LDAP search results are cached. A value of zero indicates an unlimited cache time limit.

DISTINGUISHED_NAME(block-descriptor)
specifies the location of the LDAP distinguished name, of the user permitted to bind to the chosen server. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data.

For more information on block-descriptors, see "XPI syntax" on page 393.

LDAP_BIND_PROFILE(block-descriptor)
specifies the location of the name of a RACF profile in the LDAPBIND class that contains the URL and credentials for the LDAP server being accessed. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data.

For more information on block-descriptors, see "XPI syntax" on page 393. You should specify either LDAP_BIND_PROFILE, or all three LDAP_SERVER_URL, DISTINGUISHED_NAME and PASSWORD parameters.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API, in response to receiving URL and user credentials.

LDAP_SERVER_URL(block-descriptor)
specifies the location of the LDAP URL (in the format ldap://server:port) of the LDAP server being accessed. If the colon and port number are omitted, the port defaults to 389. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data.

For more information on block-descriptors, see "XPI syntax" on page 393.

LDAP_SESSION_TOKEN(name4)
the name of the fullword token that specifies the LDAP connection.

PASSWORD(block-descriptor)
specifies the location of the password for the user identified in the DISTINGUISHED_NAME input. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data.

For more information on block-descriptors, see "XPI syntax" on page 393.

RESPONSE and REASON values for BIND_LDAP

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_BUFFER_LENGTH
	INVALID_LDAP_PROFILE
	INVALID_LDAP_URL
	LDAP_INACTIVE
	NOTAUTH
	NOTFOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The END_BROWSE_RESULTS call

The END_BROWSE_RESULTS call allows you to end the browse session that was started by the START_BROWSE_RESULTS call.

END_BROWSE_RESULTS

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(END_BROWSE_RESULTS),
    SEARCH_TOKEN(name4),]
    [OUT,
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

RESPONSE and REASON values for END_BROWSE_RESULTS

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
	INVALID_CALLING_SEQUENCE
	NOTFOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The FLUSH_LDAP_CACHE call

The FLUSH_LDAP_CACHE call removes the contents of all cached search responses for the specified LDAP connection.

FLUSH_LDAP_CACHE

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(FLUSH_LDAP_CACHE),
    LDAP_SESSION_TOKEN(name4),]
    [OUT,
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

LDAP_SESSION_TOKEN(name4)
the name of the fullword token that was returned by the BIND_LDAP function.

RESPONSE and REASON values for FLUSH_LDAP_CACHE

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
	LDAP_INACTIVE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The FREE_SEARCH_RESULTS call

The FREE_SEARCH_RESULTS call releases all storage held by the SEARCH_LDAP function. The search results are terminated and the search token is invalidated. If the application does not call the FREE_SEARCH_RESULTS function, it is invoked by CICS when the task is terminated.

FREE_SEARCH_RESULTS

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(FREE_SEARCH_RESULTS),
    SEARCH_TOKEN(name4),]
    [OUT,
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

RESPONSE and REASON values for FREE_SEARCH_RESULTS

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The GET_ATTRIBUTE_VALUE call

You can use the GET_ATTRIBUTE_VALUE call to retrieve the value associated with an attribute returned by the SEARCH_LDAP call. An entry is an LDAP record, and an attribute is one element within an entry. The attribute can be returned by either the GET_NEXT_ATTRIBUTE function, or by specifying the name of the attribute.

GET_ATTRIBUTE_VALUE

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(GET_ATTRIBUTE_VALUE),
    SEARCH_TOKEN(name4),
    LDAP_ATTRIBUTE_NAME(block-descriptor),
    LDAP_ATTRIBUTE_VALUE(buffer-descriptor),
    [ATTRIBUTE_TYPE(name4),]
    [VALUE_ARRAY_POSITION(name4),]]
    [OUT,
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

ATTRIBUTE_TYPE(name4)
Specifies the keyword CHARACTER or BINARY, indicating the format of the attribute. If this parameter is not specified, a value of CHARACTER is assumed.

LDAP_ATTRIBUTE_NAME(block-descriptor)
Specifies the location of the LDAP attribute name. The block-descriptor is two fullwords of data, in which the first word contains the address of the attribute name, and the second word contains the length in bytes of the attribute name. For more information on block-descriptors, see XPI syntax.

LDAP_ATTRIBUTE_VALUE(buffer-descriptor)
Indicates the buffer where you want the attribute value returned. A group of three fullwords are specified for the buffer-descriptor:

- The address where the result is returned.
- The maximum size in bytes, of the data returned.
- The actual length in bytes of the result. This can be specified as *, and the length is then returned in DDAP_LDAP_ATTRIBUTE_VALUE_N.

For more information on buffer-descriptors, see XPI syntax.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

VALUE_ARRAY_POSITION(name4)
Specifies the position of the requested value, in the value array for the current attribute. This parameter is only required if multiple values are expected. Array indexing starts at position 1.

RESPONSE and REASON values for GET_ATTRIBUTE_VALUE

RESPONSE	REASON
OK	None
EXCEPTION	INVALID_TOKEN
	NOTFOUND
	INVALID_BUFFER_LENGTH
	INVALID_CALLING_SEQUENCE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The GET_NEXT_ATTRIBUTE call

The GET_NEXT_ATTRIBUTE call allows you to get the next attribute in a series, from an entry returned by the SEARCH_LDAP call. An entry is an LDAP record, and an attribute is one element within an entry.

GET_NEXT_ATTRIBUTE

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(GET_NEXT_ATTRIBUTE),
    SEARCH_TOKEN(name4),
    LDAP_ATTRIBUTE_NAME(buffer-descriptor),]
    [OUT,
    [LDAP_RESPONSE(name4),]
    [VALUE_COUNT(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

LDAP_ATTRIBUTE_NAME(buffer-descriptor)
indicates the buffer where you want the attribute name returned. A group of three fullwords are specified for the buffer-descriptor:

- The address where the data is returned.
- The maximum size in bytes, of the data returned.
- The actual length in bytes of the data. This can be specified as *, and the length is then returned in DDAP_LDAP_ATTRIBUTE_NAME_N.

For more information on buffer-descriptors, see XPI syntax.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

VALUE_COUNT(name4)
a fullword containing the number of values returned for this attribute. There is usually one value returned.

RESPONSE and REASON values for GET_NEXT_ATTRIBUTE

RESPONSE	*REASON*
OK	None
EXCEPTION	BROWSE_END
	INVALID_BUFFER_LENGTH
	INVALID_CALLING_SEQUENCE
	INVALID_TOKEN
	NOT_FOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The GET_NEXT_ENTRY call

The GET_NEXT_ENTRY call allows you to get the next entry, from a series of entries returned by the SEARCH_LDAP call. An entry is an LDAP record. The distinguished name associated with the entry is returned by this call.

GET_NEXT_ENTRY

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(GET_NEXT_ENTRY),
    SEARCH_TOKEN(name4),
    [DISTINGUISHED_NAME(buffer-descriptor),]]
    [OUT,
    [LDAP_RESPONSE(name4),]
    [ATTRIBUTE_COUNT(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

ATTRIBUTE_COUNT(name4)
specifies the number of attributes in the retrieved entry.

DISTINGUISHED_NAME(buffer-descriptor)
indicates the buffer where you want the distinguished name of the next entry in the search returned. A group of three fullwords are specified for the buffer-descriptor:

- The address where the data is returned.
- The maximum size in bytes, of the data is returned.
- The actual length in bytes of the data. This can be specified as *, and the length is then returned in DDAP_DISTINGUISHED_NAME_N.

For more information on buffer-descriptors, see "XPI syntax" on page 393.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

RESPONSE and REASON values for GET_NEXT_ENTRY

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
	INVALID_BUFFER_LENGTH
	INVALID_CALLING_SEQUENCE
	BROWSE_END
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The SEARCH_LDAP call

The SEARCH_LDAP call sends a search request to a specified LDAP server. The search specifies an LDAP distinguished name, that is the target of the search.

The search returns a series of results (attributes or entries) that can be browsed or selected. An entry is an LDAP record, and an attribute is one element within an entry.

SEARCH_LDAP

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(SEARCH_LDAP),
    LDAP_SESSION_TOKEN(name4),
    DISTINGUISHED_NAME(block-descriptor),
    [FILTER(block-descriptor),]
    [SEARCH_TIME_LIMIT(name4),]]
    [OUT,
    SEARCH_TOKEN(name4),
    [LDAP_RESPONSE(name4),]
    [ENTRY_COUNT(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

DISTINGUISHED_NAME(block-descriptor)
specifies the location of the LDAP distinguished name. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data. For more information on block-descriptors, see "XPI syntax" on page 393.

ENTRY_COUNT(name4)
the number of LDAP entries returned by the search.

FILTER(block-descriptor)
specifies the location of an LDAP filter string that limits the search. If this parameter is not specified or is zero, the search filter is set to `(objectClass=*)`. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data. For more information on block-descriptors, see "XPI syntax" on page 393.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

LDAP_SESSION_TOKEN(name4)
the name of the fullword token that was returned by the BIND_LDAP function.

SEARCH_TIME_LIMIT(name4)
specifies the time limit for the search (in seconds). If the search is not successful within this time limit, the search is abandoned. If this parameter is not specified or is zero, the search time is unlimited.

SEARCH_TOKEN(name4)
the name of the fullword token that identifies and holds the current position in the search.

RESPONSE and REASON values for SEARCH_LDAP

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_BUFFER_LENGTH
	INVALID_TOKEN
	NOTFOUND
	TIMED_OUT
	LDAP_INACTIVE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The START_BROWSE_RESULTS call

The START_BROWSE_RESULTS call allows you to browse the results (attributes or entries) returned by the SEARCH_LDAP call. START_BROWSE_RESULTS starts the browse at the first or only entry returned (there may be multiple entries returned by the search). The GET_NEXT_ENTRY call allows you to retrieve other entries.

START_BROWSE_RESULTS can be issued more than once for a SEARCH_TOKEN. If the call is issued after a GET_NEXT_ENTRY or GET_NEXT_ATTRIBUTE call, the browse position will be reset to the start of the search results.

START_BROWSE_RESULTS

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(START_BROWSE_RESULTS),
    SEARCH_TOKEN(name4),
    [DISTINGUISHED_NAME(buffer-descriptor),]]
    [OUT,
    [LDAP_RESPONSE(name4),]
    [ATTRIBUTE_COUNT(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

DISTINGUISHED_NAME(buffer-descriptor)
indicates the buffer where you want the distinguished name of the first, or only located result returned. A group of three fullwords are specified for the buffer-descriptor:

- The address where the data is returned.
- The length of the buffer in bytes, where the data is returned.
- The maximum length in bytes of the data. This can be specified as *, and the length is then returned in DDAP_DISTINGUISHED_NAME_N.

For more information on buffer-descriptors, see "XPI syntax" on page 393.

ATTRIBUTE_COUNT(name4)
a fullword indicating the number of attributes that can be browsed in the current entry.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

RESPONSE and REASON values for START_BROWSE_RESULTS

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
	INVALID_BUFFER_LENGTH
	INVALID_CALLING_SEQUENCE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The UNBIND_LDAP call

The UNBIND_LDAP call terminates a session with an LDAP server.

UNBIND_LDAP

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(UNBIND_LDAP),
    LDAP_SESSION_TOKEN(name4),]
    [OUT,
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

LDAP_SESSION_TOKEN(name4)
the name of the fullword token that was returned by the BIND_LDAP function.

RESPONSE and REASON values for UNBIND_LDAP

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
	LDAP_INACTIVE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

Dispatcher XPI functions

The XPI provides six dispatcher functions. These functions are the DFHDSSRX calls ADD_SUSPEND, SUSPEND, RESUME, DELETE_SUSPEND, and WAIT_MVS, and the DFHDSATX call CHANGE_PRIORITY.

Use of these dispatcher calls is limited. Check the details supplied for each exit in Chapter 1, "Global user exit programs," on page 3 before using any functions.

Note:

1. You must issue an ADD_SUSPEND call to create a suspend token **before** you issue a SUSPEND or RESUME call.
2. If a suspended task is canceled, the SUSPEND fails with a RESPONSE value of 'PURGED' and a REASON value of 'TASK_CANCELLED'. A corresponding RESUME call returns with a RESPONSE value of 'EXCEPTION' and a REASON value of 'TASK_CANCELLED'.
3. If a suspended task is timed out, the SUSPEND fails with a RESPONSE value of 'PURGED' and a REASON value of 'TIMED_OUT'. A corresponding RESUME call returns with a RESPONSE value of 'EXCEPTION' and a REASON value of 'TIMED_OUT'.

4. Dispatcher protocols require that you issue a RESUME even if the SUSPEND was purged (due to task cancel or time out). You must issue only one RESUME for each SUSPEND call.

Synchronization protocols for SUSPEND and RESUME processing

If you use XPI SUSPEND and RESUME processing, you must observe the correct protocols, so that task purging can be handled effectively.

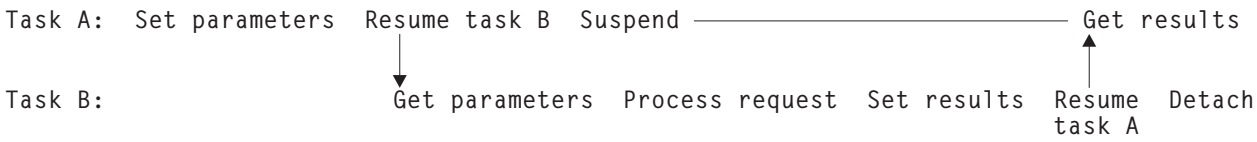
The normal synchronization protocol

In the normal case, synchronization involves two tasks and three operations.

In the following sample operations, the tasks are A (the task that requests a service) and B (the task that processes a request from task A).

1. Task A starts the request by:
   - Setting the parameters to be used by task B
   - Resuming task B
   - Issuing the SUSPEND call.
2. Task B performs the action by:
   - Getting the parameters
   - Performing the action
   - Setting the results
   - Terminating (or waiting for new work).
3. Task A ends the interaction by:
   - Getting the results left by task B.

This sequence looks like:



Ignoring the Resume and Suspend, the execution amounts to:

```
Set parameters; Get parameters; Process request; Set results; Get results
```

where these actions are always **sequential**.

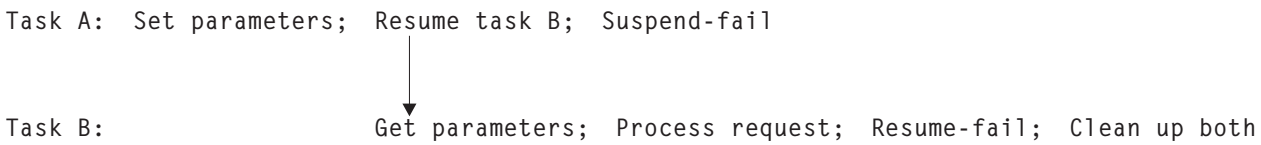
The synchronization protocol and task purge

If one of the tasks is to be purged, it is task A, because task A is the one suspended. In this case, execution of task A after the failed SUSPEND would be in parallel with task B; the proper serialization would be lost. If the program remained unchanged, Process request and Set results would be taking place at the same time as Get results, with unpredictable results.

Alternative approach to task purge

One way of preventing this problem is to ensure that task A, if it is to be purged, does not do anything that could interfere with task B. It might mean that A must not detach, if doing so releases storage that B needs to access. Because the only task that is now involved is task B, task B is left with the responsibility of cleaning up for both tasks.

The sequence is shown in the following diagram:



Because task-purging is effective only if performed between SUSPEND and RESUME, Suspend-fail precedes Resume-fail. With the same constraints on serialization as in the normal synchronization protocol, the task-purge protocol can be logically reduced to the following sequence:

```
Set parameters; Get parameters; Process request; Clean up
```

The difference is that Set results and Get results are replaced by Clean up. It is vital that only these two sequences can happen; this means that both programs must be coded correctly. CICS ensures that both tasks are told either that SUSPEND and RESUME processing worked, or that it failed.

The following shows the programming steps that conform to these rules:

```
Program for Task A                  Program for Task B
SET PARAMETERS;
RESUME B;                           GET PARAMETERS;
SUSPEND A;                          PROCESS REQUEST;
                                    RESUME A;
if                                  if
  RESPONSE = OK                       RESPONSE ¬= OK
then                                then
  GET RESULTS;                        CLEAN UP;
endif                               endif
```

If both the SUSPEND and RESUME return 'OK', the example follows the rules for the normal synchronization; processing finishes at Get results. If neither SUSPEND nor RESUME returns 'OK', the example follows the rules for the task-purge protocol, and processing finishes at Clean up.

The sequence described previously is one method for dealing with the problem of task purge. Using this method, task B does not know, when it is processing the request, whether or not task A has been purged; this means that B must take great care in its use of resources owned by A (in case A has been purged). In some situations, this restriction may cause difficulties.

A different approach is as follows; if task A is to be purged:

1. A communicates to B that it is no longer available, thus informing B not to use any resources owned by A.
2. A performs its own clean-up processing (including issuing the RESUME call for the purged SUSPEND, as required by the dispatcher protocols), and abends.

3. B performs its own clean-up processing.

The ADD_SUSPEND call

ADD_SUSPEND acquires a suspend token that can later be used to identify a SUSPEND/RESUME pair.

ADD_SUSPEND

```
DFHDSSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(ADD_SUSPEND),
      [RESOURCE_NAME(name16 | string | 'string'),]
      [RESOURCE_TYPE(name8 | string | 'string'),]]
      [OUT,
      SUSPEND_TOKEN(name4 | (Rn)),
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

RESOURCE_NAME(name16 | string | "string")
specifies a 16-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name16
The name of the location where a 16-byte value is stored.

string A string of characters without intervening blanks; if it is not 16 bytes long, it is extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: RESOURCE_NAME on ADD_SUSPEND supplies a default value which is used if RESOURCE_NAME is not specified on a SUSPEND call.

RESOURCE_TYPE(name8 | string | "string")
specifies an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name8 The name of the location where an 8-byte value is stored.

string A string of characters without intervening blanks; if it is not 8 bytes long, it is extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: RESOURCE_TYPE on ADD_SUSPEND supplies a default value which is used if RESOURCE_TYPE is not specified on a SUSPEND call.

SUSPEND_TOKEN(name4 | (Rn))
returns a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

name4 The name of a 4-byte field where the token is stored

(Rn) A register into which the token value is loaded.

RESPONSE and REASON values for ADD_SUSPEND

RESPONSE	*REASON*
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The CHANGE_PRIORITY call

CHANGE_PRIORITY allows the issuing task to change its own priority. It cannot be used to change the priority of another task. This command causes the issuing task to release control, and so provide other tasks with the opportunity to run.

CHANGE_PRIORITY

```
DFHDSATX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(CHANGE_PRIORITY),
      PRIORITY(name1 | (Rn) | decimalint | literalconst),]
      [OUT,
      [OLD_PRIORITY(name1 | (Rn)),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

OLD_PRIORITY(name1 | (Rn))
returns the previous priority of the issuing task.

name1 The name of a 1-byte field where the task's previous priority is stored

(Rn) A register in which the low-order byte receives the previous priority value and the other bytes are set to zero.

PRIORITY(name1 | (Rn) | decimalint | literalconst)
specifies the new priority to be assigned to the issuing task.

name1 The name of a 1-byte field, with a value in the range 0 through 255.

(Rn) A register with the low-order byte containing the new priority value.

decimalint
A decimal integer not exceeding 255 in value. Neither an expression nor hexadecimal notation is allowed.

literalconst
A number in the form of a literal, for example B'00000000', X'FF', X'FCF4', "0" or an equate symbol with a similar value.

RESPONSE and REASON values for CHANGE_PRIORITY

RESPONSE	*REASON*
OK	None

RESPONSE	*REASON*
DISASTER	None
INVALID	None
KERNERROR	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The DELETE_SUSPEND call

DELETE_SUSPEND releases a suspend token associated with this task.

DELETE_SUSPEND

```
DFHDSSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(DELETE_SUSPEND),
      SUSPEND_TOKEN(name4 | (Rn)),]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

SUSPEND_TOKEN(name4 | (Rn))
specifies a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

name4 The name of a 4-byte field, where the token obtained by an ADD_SUSPEND call has been stored

(Rn) A register containing the token value previously obtained.

RESPONSE and REASON values for DELETE_SUSPEND

RESPONSE	*REASON*
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The RESUME call

RESUME restarts execution of a task that is suspended or timed out.

There must be only one RESUME request for each SUSPEND. However, because this is an asynchronous interface, a SUSPEND can be received either before or after its corresponding RESUME. You must ensure that you keep account of the SUSPEND and RESUME requests issued from your exit program.

RESUME

```
DFHDSSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(RESUME),
      SUSPEND_TOKEN(name4 | (Rn)),
      [COMPLETION_CODE(name1 | (Rn)),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

COMPLETION_CODE(name1 | (Rn))
specifies a user-defined "reason for RESUME" code during suspend and resume processing.

name1 The name of a 1-byte area to receive the code

(Rn) A register, in which the low-order byte contains the completion code and the other bytes are zero.

SUSPEND_TOKEN(name4 | (Rn))
specifies a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

name4 The name of a location where you have a 4-byte token previously obtained as output from an ADD_SUSPEND call

(Rn) A register containing the token value.

RESPONSE and REASON values for RESUME

RESPONSE	*REASON*
OK	None
EXCEPTION	TASK_CANCELLED
	TIMED_OUT
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.
2. 'TASK_CANCELLED' means that the task was canceled by operator action while it was suspended, and that the suspend token is available for use.

The SUSPEND call

SUSPEND suspends execution of a running task.

Suspended tasks can be resumed in one of two ways. You can issue the XPI RESUME call, or the task is resumed automatically if the INTERVAL value that you specify on the DFHDSSRX macro expires. Suspended tasks can also be purged by the operator, or by an application, or by the deadlock timeout facility.

SUSPEND

```
DFHDSSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SUSPEND),
      PURGEABLE(YES|NO),
      SUSPEND_TOKEN(name4 | (Rn)),
      [INTERVAL(name4 | (Rn)),]
      [RESOURCE_NAME(name16 | string | 'string'),]
      [RESOURCE_TYPE(name8 | string | 'string'),]
      [TIME_UNIT(SECOND|MILLI_SECOND),]
      [WLM_WAIT_TYPE,]]
      [OUT,
      [COMPLETION_CODE(name1 | (Rn)),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

COMPLETION_CODE (name1 | (Rn))
Returns a user-defined "reason for action" code during suspend and resume processing.

name1 The name of a 1-byte area to receive the code. The value in this field is user-defined, and is ignored by CICS.

(Rn) A register in which the low-order byte contains the completion code and the other bytes are zero.

INTERVAL(name4 | (Rn))
Specifies in seconds or milliseconds the time after which the task is automatically resumed and given a RESPONSE value of PURGED and a REASON value of TIMED_OUT. The time unit used on the INTERVAL option depends on the setting of the TIME_UNIT option. The INTERVAL value overrides any timeout (DTIMOUT) value specified for the transaction.

name4 The name of a 4-byte area, which is interpreted as a binary fullword.

(Rn) A register containing the interval value, a binary fullword.

PURGEABLE(YES|NO)
Specifies whether your code can cope with the request being abnormally terminated as a result of a purge. There are four types of purge, as shown in Table 42. Specifying PURGEABLE(NO) tells the dispatcher:

- To reject any attempt to PURGE the task.
- To suppress the deadlock timeout (DTIMOUT) facility (if applicable to this task) for the duration of this request.

Table 42. SUSPEND call - RESPONSE(PURGED)

REASON	CONDITION	PURGEABLE (NO)	PURGEABLE (YES)
TASK_CANCELLED	PURGE	Canceled	Proceeds normally
	FORCEPURGE	Proceeds normally	Proceeds normally
TIMED_OUT	DTIMOUT	Canceled	Proceeds normally
	INTERVAL	Proceeds normally	Proceeds normally

Note: A FORCEPURGE always assumes that the user wants the task to be purged, and so overrides the PURGEABLE(NO) option. If the user has set an INTERVAL, then this, too, overrides the PURGEABLE(NO) option.

RESOURCE_NAME(name16 | string | "string")

Specifies a 16-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name16
The name of the location where a 16-byte value is stored.

string A string of characters without intervening blanks; if it is not 16 bytes long, it is extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note:

1. CICS does not use the RESOURCE_NAME information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are described in The resources on which CICS system tasks can wait in Troubleshooting.
2. If RESOURCE_NAME is not specified, the default value, if any, from ADD_SUSPEND is used.

RESOURCE_TYPE(name8 | string | "string")

Specifies an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name8 The name of the location where an 8-byte value is stored.

string A string of characters without intervening blanks; if it is not 8 bytes long, it is extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note:

1. CICS does not use the RESOURCE_TYPE information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are documented in The resources on which CICS system tasks can wait in Troubleshooting.
2. If RESOURCE_TYPE is not specified, the default value, if any, from ADD_SUSPEND is used.

SUSPEND_TOKEN(name4 | (Rn))

Specifies a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

name4 The name of a location where you have a 4-byte token previously obtained as output from an ADD_SUSPEND call

(Rn) A register containing the token value.

TIME_UNIT(SECOND | MILLI_SECOND)
Specifies the time unit used on the INTERVAL option.

SECOND
The INTERVAL option specifies the number of seconds before timeout.

MILLI_SECOND
The INTERVAL option specifies the number of milliseconds before timeout.

WLM_WAIT_TYPE(name1)
Specifies, in a 1-byte location, the reason for suspending the task. This reason indicates the nature of the wait state to the MVS workload manager.

The equated values for the type of wait are as follows:

CMDRESP
Waiting on a command response.

CONV
Waiting on a conversation.

DISTRIB
Waiting on a distributed request.

IDLE
A CICS task, acting as a work manager, that has no work request that is allowed to service within the monitoring environment. For example, journaling code that suspends itself when there are no journaling I/O operations to perform.

IO Waiting on an I/O operation or indeterminate I/O-related operation (locks, buffer, string, and so on).

LOCK
Waiting on a lock.

MISC
Waiting on an unidentified resource.

Note: This value is the default reason given to the wait if you suspend a task and do not specify the WLM_WAIT_TYPE parameter.

OTHER_PRODUCT
Waiting on another product to complete its function; for example, when the workload has been passed to DB2.

SESS_LOCALMVS
Waiting on the establishment of a session in the MVS image on which this CICS region is running.

SESS_NETWORK
Waiting on the establishment of a session elsewhere in the network (that is, not on this MVS image).

SESS_SYSPLEX
Waiting on establishment of a session somewhere in the sysplex (that is, not on this MVS image).

TIMER
Waiting on the timeout of a timer (for example, a task that puts itself to sleep).

If you are running CICS in an MVS goal-mode workload management environment (that is, you are using goal-oriented performance management), specify the reason for suspending the task on the WLM_WAIT_TYPE parameter.

Table 43. RESPONSE and REASON values for SUSPEND

RESPONSE	REASON
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	TASK_CANCELLED
	TIMED_OUT

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.
2. TASK_CANCELLED means that the task has been canceled by operator action or by an application command.
3. After a PURGED response, the suspend token must not be reused in another SUSPEND until it has been reset by a RESUME corresponding to the purged SUSPEND.
4. TIMED_OUT means that the task has been automatically resumed because the specified INTERVAL (or the timeout value specified at task attach) has expired. The token, however, remains suspended and must be the object of a RESUME before it can be the object of a DELETE_SUSPEND.

The WAIT_MVS call

WAIT_MVS requests a wait on an MVS event control block (ECB) or on a list of MVS ECBs. For example, you could issue the WAIT_MVS to wait for completion of an MVS task for which you have issued ATTACH and provided a task-completion ECB.

The dispatcher does not clear the ECBs when a WAIT_MVS request is received. If any ECB is already posted, control is returned immediately to the exit program with a response of 'OK'.

A single ECB must not be the subject of more than one wait at a time. If any ECB is already being waited on when a WAIT_MVS request is received, the request is rejected. The RESPONSE code is 'DSSR_INVALID', and the REASON code 'DSSR_ALREADY_WAITING'.

Note: ECBs used in WAIT_MVS requests must always be posted using the MVS POST macro.

WAIT_MVS

```
DFHDSSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(WAIT_MVS),
      {ECB_ADDRESS(name4 | (Ra)) | ECB_LIST_ADDRESS(name4 | (Ra)),}
      PURGEABLE(YES|NO),
      [INTERVAL(name4 | (Rn)),]
```

```
[RESOURCE_NAME(name16 | string | 'string'),]
[RESOURCE_TYPE(name8 | string | 'string'),]]
[TIME_UNIT(SECOND|MILLI_SECOND),]
[WLM_WAIT_TYPE,]
[OUT,
RESPONSE(name1 | *),
REASON(name1 | *)]
```

This command is threadsafe.

ECB_ADDRESS(name4 | (Ra))
Specifies the address of the ECB to be waited on.

name4 The name of a location that contains an ECB address.

(Ra) A register that contains the address of an ECB.

ECB_LIST_ADDRESS(name4 | (Ra))
Specifies the address of a list of ECB addresses to be waited on.

name4 The name of a location that contains an ECB address, possibly followed by more ECB addresses. The last address word in the list has the high-order bit set to 1.

(Ra) A register pointing to an address list as previously described.

INTERVAL(name4 | (Rn))
Specifies in seconds or milliseconds the time after which the task is automatically resumed and given a RESPONSE value of 'PURGED' and a REASON value of 'TIMED_OUT'. The time unit used on the INTERVAL option depends on the setting of the TIME_UNIT option.

The INTERVAL value overrides any time-out (DTIMOUT) value specified for the transaction.

name4 The name of a 4-byte area, which is interpreted as a binary fullword

(Rn) A register containing the interval value, a binary fullword.

PURGEABLE(YES|NO)
Specifies whether your code can cope with the request being abnormally terminated as a result of a purge. There are four types of purge, as shown in Table 44. Specifying PURGEABLE(NO) tells the dispatcher:

- To reject any attempt to PURGE the task
- To suppress the deadlock timeout (DTIMOUT) facility (if applicable to this task) for the duration of this request.

Table 44. WAIT_MVS call - RESPONSE(PURGED)

REASON	CONDITION	PURGEABLE (NO)	PURGEABLE (YES)
TASK_CANCELLED	PURGE	Canceled	Proceeds normally
	FORCEPURGE	Proceeds normally	Proceeds normally
TIMED_OUT	DTIMOUT	Canceled	Proceeds normally
	INTERVAL	Proceeds normally	Proceeds normally

Note: A FORCEPURGE always assumes that the user wants the task to be purged, and so overrides the PURGEABLE(NO) option. If the user has set an INTERVAL, then this, too, overrides the PURGEABLE(NO) option.

RESOURCE_NAME(name16 | string | "string")
Specifies a 16-character string that can be used to document and trace the

resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name16
The name of the location where a 16-byte value is stored.

string A string of characters without intervening blanks; if it is not 16 bytes long, it is extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: CICS does not use the RESOURCE_NAME information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are documented in The resources on which CICS system tasks can wait in Troubleshooting.

RESOURCE_TYPE(name8 | string | "string")
Specifies an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name The name of the location where an 8-byte value is stored.

string A string of characters without intervening blanks; if it is not 8 bytes long, it will be extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: CICS does not use the RESOURCE_TYPE information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are documented in The resources on which CICS system tasks can wait in Troubleshooting.

TIME_UNIT(SECOND | MILLI_SECOND)
Specifies the time unit used on the INTERVAL option.

SECOND
The INTERVAL option specifies the number of seconds before timeout.

MILLI_SECOND
The INTERVAL option specifies the number of milliseconds before timeout.

WLM_WAIT_TYPE(name1)
Specifies, in a 1-byte location, the reason for suspending the task. This indicates the nature of the task's wait state to the MVS workload manager.

The equated values for the type of wait are as follows:

CMDRESP
Waiting on a command response.

CONV
Waiting on a conversation.

DISTRIB
Waiting on a distributed request.

IDLE
A CICS task, acting as a work manager, that has no work request that is allowed to service within the monitoring environment. For example, journaling code that suspends itself when there are no journaling I/O operations to perform.

IO Waiting on an I/O operation or indeterminate I/O-related operation (locks, buffer, string, and so on).

LOCK
Waiting on a lock.

MISC
Waiting on an unidentified resource. This is the default reason given to the wait if you suspend a task and do not specify the WLM_WAIT_TYPE parameter.

OTHER_PRODUCT
Waiting on another product to complete its function; for example, when the workload has been passed to DB2.

SESS_LOCALMVS
Waiting on the establishment of a session in the MVS image on which this CICS region is running.

SESS_NETWORK
Waiting on the establishment of a session elsewhere in the network (that is, not on this MVS image).

SESS_SYSPLEX
Waiting on establishment of a session somewhere in the sysplex (that is, not on this MVS image).

TIMER
Waiting on the timeout of a timer (for example, a task that puts itself to sleep).

If you are running CICS in an MVS goal-mode workload management environment (that is, you are using goal-oriented performance management), you are recommended to specify the reason for suspending the task on the WLM_WAIT_TYPE parameter.

Table 45. RESPONSE and REASON values for WAIT_MVS

RESPONSE	REASON
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	TASK_CANCELLED TIMED_OUT

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.
2. TIMED_OUT is returned if the INTERVAL expires, or if a deadlock timeout interval expires.
3. TASK_CANCELLED means that the task has been canceled by operator action or by an application command.

Dump control XPI functions

The XPI provides two dump control functions. These are the DFHDUDUX macro calls SYSTEM_DUMP and TRANSACTION_DUMP.

Restriction: DFHDUDUX calls cannot be used in any exit program invoked from any global user exit point in the:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Transient data program.

The SYSTEM_DUMP call

SYSTEM_DUMP causes a system dump to be taken. If the system dump code that you supply on input is in the system dump code table, the dump may be suppressed.

For information about the dump table and how it works, refer to the *CICS Problem Determination Guide* and SET SYSDUMPCODE, in the *CICS System Programming Reference* manual.

SYSTEM_DUMP

```
DFHDUDUX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SYSTEM_DUMP),
      SYSTEM_DUMPCODE(name8 | string | "string"),
      [CALLER(block-descriptor),]
      [TITLE(block-descriptor),]]
      [OUT,
      DUMPID(name9 | *),
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

CALLER(block-descriptor)
specifies the source of a system dump request. The information that you supply here appears in the dump header, so you could use it to identify the exit program that initiated the system dump request. For a description of valid block-descriptors, see block-descriptor.

DUMPID(name9 | *)
returns the dump identifier.

name9 The name of a 9-byte field to receive the assigned ID.

SYSTEM_DUMPCODE(name8 | string | "string")
specifies the code corresponding to the error that caused this system dump

call. System dump codes are held in the dump table; for information about the dump table and how it works, refer to the *CICS Problem Determination Guide* and SET SYSDUMPCODE, in the *CICS System Programming Reference* manual.

name8 The name of a location containing an 8-byte string.

string A string of characters without intervening blanks. The macro generates, from the string, a literal constant of length 8 bytes, extending with blanks or truncating as required.

"string"
A string, enclosed in quotation marks and possibly containing blanks. This value is processed in the same way as the preceding "string".

TITLE(block-descriptor)
specifies an area containing the text you want to appear in the dump header when the system dump is printed.

RESPONSE and REASON values for SYSTEM_DUMP

RESPONSE	*REASON*
OK	None
EXCEPTION	FESTAE_FAILED
	INSUFFICIENT_STORAGE
	IWMWQWRK_FAILED
	NO_DATASET
	PARTIAL_SYSTEM_DUMP
	SDUMP_BUSY
	SDUMP_FAILED
	SDUMP_NOT_AUTHORIZED
	SUPPRESSED_BY_DUMPOPTION
	SUPPRESSED_BY_DUMPTABLE
	SUPPRESSED_BY_USEREXIT
DISASTER	None
INVALID	INVALID_DUMPCODE
	INVALID_PROBDESC
	INVALID_SVC_CALL
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The TRANSACTION_DUMP call

TRANSACTION_DUMP causes a transaction dump to be taken. If the transaction dump code that you supply on input is in the transaction dump code table, the dump may be suppressed and, optionally, a system dump may be taken.

For information about the dump table and how it works, refer to the *CICS Problem Determination Guide* and SET TRANDUMPCODE, in the *CICS System Programming Reference* manual.

Valid characters include uppercase characters (A-Z), lowercase characters (a-z), digits (0-9), and the special characters $ @ # / % & ? ! : | ; , ¢ + * ¬ - and _. In some cases, the characters < > . = and " are also valid depending on where you set them. Any lowercase characters you enter are converted to uppercase.

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Chapter 5, "Writing initialization and shutdown programs," on page 409.

TRANSACTION_DUMP

```
DFHDUDUX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(TRANSACTION_DUMP),
      TRANSACTION_DUMPCODE(name4 | string | 'string')
      [CSA(NO|YES),]
      [PROGRAM(NO|YES),]
      [SEGMENT(block-descriptor),]
      [SEGMENT_LIST(block-descriptor),]
      [TCA(NO|YES),]
      [TERMINAL(NO|YES),]
      [TRANSACTION(NO|YES),]
      [TRT(NO|YES),]]
      [OUT,
      DUMPID(name9 | *),
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

Note: This command is **NOT** threadsafe.

CSA(NO|YES)
specifies whether the common system area (CSA) is to be included in the transaction dump. The default is NO.

DUMPID(name9 | *)
returns the dump identifier.

name9 The name of a 9-byte field to receive the assigned ID.

PROGRAM(NO|YES)
specifies whether all program storage areas associated with this task are to be included in the transaction dump. The default is NO.

SEGMENT(block-descriptor)
specifies the address and the length of a single block of storage that is to be dumped. See block-descriptor for a description of valid block-descriptors. SEGMENT and SEGMENT_LIST are mutually exclusive.

SEGMENT_LIST(block-descriptor)
specifies the address and length of a *set* of contiguous word pairs. The first word in each pair specifies the **length** in bytes of a storage segment to be dumped; the second word contains the **address** of the storage segment. The end of the list must be marked by a word containing X'FFFFFFFF'. SEGMENT and SEGMENT_LIST are mutually exclusive.

TCA(NO|YES)
specifies whether the task control area (TCA) is to be included in the transaction dump. The default is NO.

TERMINAL(NO|YES)
specifies whether all terminal storage areas associated with the task are to be included in the transaction dump. The default is NO.

TRANSACTION(NO|YES)
specifies whether all transaction storage areas associated with the task are to be included in the transaction dump. The default is NO.

TRANSACTION_DUMPCODE(name4 | string | "string")
specifies the code corresponding to the error that caused this transaction dump call. Transaction dump codes are held in the dump table; for information about the dump table and how it works, refer to the *CICS Problem Determination Guide* and SET TRANDUMPCODE, in the *CICS System Programming Reference* manual.

name4 The name of a location containing a 4-byte string.

string A string of characters without intervening blanks. The macro generates a literal constant of length 4 bytes from the string, extending with blanks or truncating as required.

"string"
A string, enclosed in quotation marks and possibly containing blanks. This value is processed in the same way as the preceding "string".

TRT(NO|YES)
specifies whether the trace table (TRT) is to be included in the transaction dump. The default is NO.

RESPONSE and REASON values for TRANSACTION_DUMP

RESPONSE	*REASON*
OK	None
EXCEPTION	FESTAE_FAILED
	INSUFFICIENT_STORAGE
	IWMWQWRK_FAILED
	NOT_OPEN
	OPEN_ERROR
	PARTIAL_SYSTEM_DUMP
	PARTIAL_TRANSACTION_DUMP
	SDUMP_BUSY
	SDUMP_FAILED
	SDUMP_NOT_AUTHORIZED
	SUPPRESSED_BY_DUMPOPTION
	SUPPRESSED_BY_DUMPTABLE
	SUPPRESSED_BY_USEREXIT
DISASTER	None
INVALID	INVALID_DUMPCODE
	INVALID_PROBDESC
	INVALID_SVC_CALL
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.
2. 'NOT_OPEN' means that the CICS dump data set is closed.
3. 'OPEN_ERROR' means that an error occurred while a CICS dump data set was being opened.
4. 'PARTIAL' means that the transaction dump resulting from this request is incomplete.

Enqueue domain XPI functions

The XPI provides two enqueue domain functions. These are the DFHNQEDX calls DEQUEUE and ENQUEUE.

The DEQUEUE function

The DEQUEUE function is provided on the DFHNQEDX macro call. It releases a resource previously enqueued by an ENQUEUE function call.

DEQUEUE

```
DFHNQEDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(DEQUEUE),
      {ENQUEUE_TOKEN(name4),|
      ENQUEUE_NAME1(address,length),[ENQUEUE_NAME2(address,length),]}
      MAX_LIFETIME(DISPATCHER_TASK),]
     [ENQUEUE_TYPE (XPI | EXECSTRN | EXECADDR),]
      [OUT,
      RESPONSE (name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

The ENQUEUE_TOKEN, ENQUEUE_NAME1, ENQUEUE_NAME2, MAX_LIFETIME (DISPATCHER_TASK), and ENQUEUE_TYPE (XPI | EXECSTRN | EXECADDR) parameters are the same as in the ENQUEUE function call.

RESPONSE and REASON values for DEQUEUE

RESPONSE	*REASON*
OK	None
EXCEPTION	
	ENQUEUE_NOT_OWNED
	ENQUEUE_LOCKED

The ENQUEUE function

The ENQUEUE function is provided on the DFHNQEDX macro call. It allows you to enqueue on a named resource.

By default, all enqueues created by XPI ENQUEUE commands are allocated to a specific enqueue pool called DISPATCH and are treated as internal to CICS. XPI enqueues do not conflict with enqueues created by **EXEC CICS ENQ** commands, which are added to different enqueue pools, depending on the enqueue model specified. For example, an active EXEC CICS ENQ on a string does not prevent an XPI ENQUEUE command on the same string being obtained.

Note:
- XPI enqueues cannot be browsed using the CICS SPI.
- XPI enqueues cannot be controlled by the use of ENQMODELs.

When you use the optional **ENQUEUE_TYPE** parameter, the XPI ENQUEUE command can enqueue on the same resource being enqueued on by EXEC CICS ENQ or vice versa. Applications can synchronize processes using EXEC CICS and EXEC XPI commands.

ENQUEUE

```
DFHNQEDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(ENQUEUE),
      ENQUEUE_NAME1(address,length),
      [ENQUEUE_NAME2(address,length),]
      MAX_LIFETIME(DISPATCHER_TASK),
     [ENQUEUE_TYPE (XPI | EXECSTRN | EXECADDR),]
      [WAIT(YES|NO),]
      [PURGEABLE(YES|NO),]]
      [OUT,
      [ENQUEUE_TOKEN(name4),]
      [DUPLICATE_REQUEST,]
      RESPONSE (name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

DUPLICATE_REQUEST
Indicates that the requesting dispatcher task already owns the resource being enqueued.

ENQUEUE_NAME1(address,length)
Specifies the high-order part of the name to be enqueued.

ENQUEUE_NAME2(address,length)
Specifies the low-order part, if any, of the name to be enqueued.

ENQUEUE_TOKEN(name4)
Enables a subsequent DEQUEUE request to identify the resource by a token rather than enqueue name, allowing the NQ domain to locate the enqueue control block directly, and hence more efficiently.

ENQUEUE_TYPE (XPI | EXECSTRN | EXECADDR)
Specifies the type of resource being enqueued on. The XPI option specifies the typical DFHNQEDX behavior. The resource pool used is exclusive to XPI and cannot be accessed by the CICS API. Use EXECSTRN or EXECADDR to indicate that ENQUEUE_NAME1 specifies an enqueue resource, located in the same namespace as the one being used by EXEC CICS ENQ. For more information about EXECSTRN and EXECADDR, see The resources on which CICS system tasks can wait in Troubleshooting.

MAX_LIFETIME(DISPATCHER_TASK)
MAX_LIFETIME(DISPATCHER_TASK) is required and specifies that all XPI enqueues are owned by the requesting dispatcher task.

If you use the ENQUEUE XPI call to ensure that your global user exit programs are threadsafe, you are recommended to free (dequeue) resources during the invocation of the global user exit program in which they were enqueued. However, because no recovery services are provided for stopping global user exits, CICS ensures that any outstanding XPI enqueues are dequeued automatically when the dispatcher task ends. If the dispatcher task is running a CICS transaction, the dispatcher task ends when the CICS transaction ends, whether normally or abnormally.

Usually, enqueues are owned by the requesting transaction, which contains units of work (UOWs), and these are used to anchor the enqueue control blocks. The XPI, however, does not require a transaction environment, and global user exits can be called under dispatcher tasks that have no transactions or UOWs.

PURGEABLE(YES|NO)
Specifies whether a purge (or timeout) request against the task is to be honored if the requesting dispatcher task has to wait for the enqueue.

WAIT(YES|NO)
Specifies whether the dispatcher task is to wait if the resource is currently enqueued to another dispatcher task.

RESPONSE and REASON values for ENQUEUE

RESPONSE	REASON
OK	None
EXCEPTION	
	ENQUEUE_BUSY
	ENQUEUE_LOCKED
	ENQUEUE_DISABLED
	LIMIT_EXCEEDED
	SYSENQ_FAILURE
PURGED	
	TASK_CANCELLED
	TIMED_OUT

Kernel domain XPI functions

The XPI provides two kernel domain functions. These are the DFHKEDSX calls START_PURGE_PROTECTION and STOP_PURGE_PROTECTION.

The START_PURGE_PROTECTION function

The START_PURGE_PROTECTION function is provided on the DFHKEDSX macro call. It inhibits purge, but not force-purge, for the current task. This function can be used by all global user exit programs to inhibit purge during a global user exit call.

In general, each START_PURGE_PROTECTION call should have a corresponding STOP_PURGE_PROTECTION function call to end the purge protection period on completion of any program logic that needs such protection.

START_PURGE_PROTECTION

```
DFHKEDSX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(START_PURGE_PROTECTION),]
       [OUT,
       RESPONSE (name1 | *)]
```

This command is threadsafe.

There are no input or output parameters on this call, only a RESPONSE.

The STOP_PURGE_PROTECTION function

The STOP_PURGE_PROTECTION function is provided on the DFHKEDSX macro call. It is re-enables purge for the current task after purge has been suspended by a preceding START_PURGE_PROTECTION function call.

STOP_PURGE_PROTECTION

```
DFHKEDSX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(STOP_PURGE_PROTECTION),]
       [OUT,
       RESPONSE (name1 | *)]
```

This command is threadsafe.

There are no input or output parameters on this call, only a RESPONSE.

Nesting purge protection calls

The START_ and STOP_PURGE_PROTECTION functions can be nested. You should ensure that, if multiple START_PURGE_PROTECTION calls are issued for a task, that the correct number of STOP_PURGE_PROTECTION calls are issued to cancel the purge protection.

If you issue two starts and only one stop, purge protection remains on for the current task.

For example, for any current task, more than one global user exit program may be driven. You must design your exit programs to ensure that purge protection is correctly cancelled. An example of nesting is shown as follows:

```
XEIIN:
 EXIT_PROG1: Calls START_PURGE_PROTECTION

   XFCREQ:
    EXIT_PROG2: Calls START_PURGE_PROTECTION

   XFCREQC:
    EXIT_PROG3: Calls STOP_PURGE_PROTECTION

XEIOUT:
 EXIT_PROG4: Calls STOP_PURGE_PROTECTION
```

Loader XPI functions

The XPI provides five loader functions. These functions are the DFHLDLDX calls ACQUIRE_PROGRAM, DEFINE_PROGRAM, DELETE_PROGRAM, IDENTIFY_PROGRAM, and RELEASE_PROGRAM.

The CICS loader services, including the XPI, recognize non-Language Environment (LE) assembler programs that are linked AMODE(64). The addressing mode of the module is shown in the returned entry point parameter. AMODE(64) is indicated when bit 0 is 0 and bit 31 is 1 (the same addressing mode convention that is used in the z/OS operating system).

Restriction: DFHLDLDX calls cannot be used in any exit program invoked from any global user exit point in the following domains or program:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Transient data program

The ACQUIRE_PROGRAM call

ACQUIRE_PROGRAM returns the entry and load point addresses, the length, and a new program token for a usable copy of the named program, which can be identified by either its name or a program token.

ACQUIRE_PROGRAM

```
DFHLDLDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(ACQUIRE_PROGRAM),
      {PROGRAM_NAME(name8 | string | 'string')|
      PROGRAM_TOKEN(name8)},
      [SUSPEND(NO|YES),]]
      [OUT,
      ENTRY_POINT(name4 | (Ra)),
      [LOAD_POINT(name4 | (Ra)),]
      [NEW_PROGRAM_TOKEN(name8),]
      [PROGRAM_ATTRIBUTE(name1 | (Rn)),]
      [PROGRAM_LENGTH(name4 | (Rn)),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

ENTRY_POINT(name4 | (Ra))
Returns the program's entry point address.

name4 The name of a 4-byte location to receive the 31-bit entry address

(Ra) A register to receive the entry address.

LOAD_POINT(name4 | (Ra))
Returns the program's load point address.

name4 The name of a 4-byte location to receive the loaded address

(Ra) A register that is to contain the load address.

NEW_PROGRAM_TOKEN(name8)
Returns the new program token for a usable copy of the named program.

name8 The name of a location to receive the 8-byte token that identifies this program and instance.

PROGRAM_ATTRIBUTE(name1 | (Rn))
Returns the program attribute.

name1 The name of a 1-byte location to receive the program attribute.

(Rn) A register in which the low-order byte receives the program attribute and the other bytes are set to zero. It can have the values RELOAD, RESIDENT, REUSABLE, or TRANSIENT.

RELOAD
The program is not reusable, and therefore several copies of the program may be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

RESIDENT
There is a single copy of the program that is not removed from storage unless deleted. RESIDENT programs must be at least quasireentrant. Any program of PROGRAM_TYPE SHARED

has the RESIDENT attribute by default. The DELETE_PROGRAM call has no effect on this type of RESIDENT program.

REUSABLE
Similar to RESIDENT, except that a REUSABLE program that is not in use can be removed from storage by CICS, for storage optimization reasons.

TRANSIENT
Similar to RESIDENT, except that a TRANSIENT program is removed from storage as soon as its use count drops to zero.

PROGRAM_LENGTH(name4 | (Rn))
Returns the length of the named program.

name4 The name of a 4-byte location that is to receive the length in bytes, expressed in binary

(Rn) A register to contain the length in bytes, expressed in binary.

PROGRAM_NAME(name8 | string | "string")
Specifies the name of the program to be acquired.

name8 The name of a location containing an 8-byte program name.

string A string of characters naming the program.

"string"
A string in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name8),
Specifies a token identifying the program whose details are to be acquired.

name8 The name of a location containing the 8-byte token obtained from a previous DEFINE_PROGRAM or ACQUIRE_PROGRAM call.

SUSPEND(NO|YES)
Specifies whether execution is to be suspended until the request can be granted.

RESPONSE and REASON values for ACQUIRE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	NO_STORAGE
	PROGRAM_NOT_DEFINED
	PROGRAM_NOT_FOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.
2. A REASON of 'NO_STORAGE' with a RESPONSE of 'EXCEPTION' means that there was insufficient storage to satisfy this request, and SUSPEND(NO) was specified.

3. A REASON of 'PROGRAM_NOT_FOUND' is returned if the program has not been included in the library concatenation, or if the link-edit failed. In such a case, the program is marked as "not executable"; it must be re-linked before it can be successfully acquired.

The DEFINE_PROGRAM call

You can use the DEFINE_PROGRAM call to define new programs to the loader domain, or to change the details of programs that are already defined. The details that you provide are recorded on the local catalog, and become available immediately. They are used on all subsequent ACQUIRE requests for the named program.

Program definitions made using DEFINE_PROGRAM are not retained over an XRF takeover. Also, the CSD is not updated, only the loader domain definitions.

DEFINE_PROGRAM

```
DFHLDLDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(DEFINE_PROGRAM),
      PROGRAM_NAME(name8 | string | 'string' ),
      [EXECUTION_KEY(CICS|USER),]
      [PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT),]
      [PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY),]
      [REQUIRED_AMODE(24|31|AMODE_ANY|64),]
      [REQUIRED_RMODE(24|RMODE_ANY),]]
      [OUT,
      [NEW_PROGRAM_TOKEN(name8),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

EXECUTION_KEY(CICS|USER)
Specifies, in conjunction with other program attributes, the type of dynamic storage area (DSA) into which the loader is to load the program.

CICS For non-reentrant programs, the program is to be loaded into a CICS DSA above or below the 16 MB line; that is, the CDSA or ECDSA. The choice of CICS DSA depends on the residence mode (RMODE) attribute of the program, as defined to the linkage-editor.

For reentrant RMODE(24) programs, the program is to be loaded into the CDSA.

USER For non-reentrant programs, the program is to be loaded into a user DSA above or below the 16 MB line; that is, the UDSA or EUDSA. The choice of user DSA depends on the residence mode (RMODE) attribute of the program, as defined to the linkage-editor.

For reentrant RMODE(24) programs, the program is to be loaded into the UDSA.

Reentrant programs eligible to reside above the 16 MB line: If a program is link-edited as reentrant with AMODE(31),RMODE(ANY), the EXECUTION_KEY option is ignored, and it is loaded into a read-only DSA (the RDSA or ERDSA). For details of the type of storage allocated for the ERDSA, see the RENTPGM system initialization parameter.

See Table 46 on page 801 for a summary of the effect of the EXECUTION_KEY option in conjunction with other factors.

Table 46. Summary of attributes defining DSA eligibility

EXECUTION_KEY option	Reentrant	Above or below 16 MB line	Dynamic storage area (DSA)
CICS	No	Below	CDSA
CICS	Yes	Below	RDSA
CICS	No	Above	ECDSA
CICS	Yes	Above	ERDSA
USER	No	Below	UDSA
USER	Yes	Below	RDSA
USER	No	Above	EUDSA
USER	Yes	Above	ERDSA

NEW_PROGRAM_TOKEN(name8)
Returns the token supplied for the newly-defined program.

name8 The name of a location to contain the 8-byte token obtained.

PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT)
Specifies the residency status of the program.

RELOAD
Every ACQUIRE_PROGRAM request for this program is satisfied by loading a new copy into storage. When a RELEASE request is issued for a copy, it is removed from storage.

Note: Do not use this attribute when defining an exit program.

RESIDENT
There is a single copy of the program that is not removed from storage unless it is deleted. RESIDENT programs must be at least quasi-reentrant.

REUSABLE
The program is at least quasi-reentrant; a single copy in storage can be used by several tasks in the system. A REUSABLE program becomes eligible for removal from storage as part of the normal dynamic program storage compression (DPSC) scheme when its use count reaches zero.

TRANSIENT
Similar to REUSABLE, except that the program is removed from storage immediately when its use count reaches zero. Specify this option only for less-frequently used programs, or for programs in systems that are critically short on storage.

PROGRAM_NAME(name8 | string | "string")
Specifies the name of the program to be defined.

name8 The name of a location where there is an 8-byte program name.

string A string of characters, without intervening blanks, naming the program.

"string"
A string of characters within quotation marks. The string length is set to 8 by padding with blanks or by truncation.

PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY)
Specifies where to load the program from.

PRIVATE
The program is in the DFHRPL or dynamic LIBRARY concatenation. A PRIVATE program need not be reentrant, and is given only limited protection from unauthorized overwriting. The degree of protection depends on the type of dynamic storage area (DSA) into which the program is loaded (see the EXECUTION_KEY option):

DSA Protection from unauthorized overwriting

CDSA Cannot be overwritten by USER tasks.

ECDSA
Cannot be overwritten by USER tasks.

ERDSA
Complete: cannot be overwritten by USER tasks or CICS tasks.

EUDSA
None.

RDSA Complete: cannot be overwritten by USER tasks or CICS tasks.

UDSA None.

SHARED
The program is located in the link pack area (LPA), is reentrant, and is protected.

TYPE_ANY
Either the copy in DFHRPL or dynamic LIBRARY concatenation, or the LPA copy of the program can be used, though preference is given to the LPA copy.

REQUIRED_AMODE(24|31|AMODE_ANY|64)
Specifies the addressing mode of the program. If, during subsequent ACQUIRE_PROGRAM processing, no copy of the program that meets the defined addressing requirement can be found, the ACQUIRE_PROGRAM call receives an EXCEPTION response and the REASON value PROGRAM_NOT_FOUND.

Note:

1. AMODE_ANY and AMODE 31 have identical meanings for this function.
2. You cannot use this option to override the link-edited addressing mode of the program.

REQUIRED_RMODE(24|RMODE_ANY)
Specifies the residency mode of the program. If, during subsequent ACQUIRE_PROGRAM processing, no copy of the program that meets the defined addressing requirement can be found, the ACQUIRE_PROGRAM call receives an EXCEPTION response and the REASON value PROGRAM_NOT_FOUND.

Note: You cannot use this option to override the link-edited residence mode of the program.

RESPONSE and REASON values for DEFINE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	CATALOG_ERROR CATALOG_NOT_OPERATIONAL
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The DELETE_PROGRAM call

DELETE_PROGRAM removes the definition of a named program from the catalog and from the list of current programs. When this request executes successfully, subsequent ACQUIRE_PROGRAM requests fail with a REASON value of 'PROGRAM_NOT_DEFINED'.

DELETE_PROGRAM

```
DFHLDLDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(DELETE_PROGRAM),
      PROGRAM_NAME(name8 | string | 'string' ),]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

PROGRAM_NAME(name8 | string | "string")
specifies the name of the program to be deleted.

name8 The name of a location containing an 8-byte program name.

string A string of characters naming the program.

"string"
A string in quotation marks. The string length is set to 8 by padding with blanks or truncating.

RESPONSE and REASON values for DELETE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The IDENTIFY_PROGRAM call

IDENTIFY_PROGRAM locates the program that is associated with an address. If the address is in a CICS-defined program, the call returns information about that program.

If the address is not associated with a loader domain CICS-defined program, the request fails with a REASON value of INSTANCE_NOT_FOUND.

IDENTIFY_PROGRAM

```
IDENTIFY_PROGRAM
DFHLDLDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(IDENTIFY_PROGRAM),
      ADDRESS(name4 | (Rn)| *),]
      [OUT,
      [PROGRAM_NAME(name8 | *),]
      [PROGRAM_ATTRIBUTE(name1 | (Rn) | *),]
      [PROGRAM_LENGTH(name4 | (Rn) | *),]
      [LOAD_POINT(name4 | (Ra) | *),]
      [ENTRY_POINT(name4 | (Ra) | *),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

ADDRESS(name4 | (Rn) | *)
The storage address that is used to identify the program.

name4 The name of a 4-byte fullword where the storage address is stored.

(Rn) A register that is set to the storage address.

PROGRAM_NAME(name8 | *)
Returns the name of the program that contains the storage address. The PROGRAM_NAME corresponds to the CICS-defined program name, not a CSECT.

name8 The name of a location to contain an 8-byte program name.

PROGRAM_ATTRIBUTE(name1 | (Rn) | *)
Returns the program attribute.

name1 The name of a 1-byte location to receive the program attribute.

(Rn) A register in which the low-order byte receives the program attribute and the other bytes are set to zero. The register can have the values RELOAD, RESIDENT, REUSABLE, or TRANSIENT.

RELOAD
The program is not reusable, and therefore several copies of the program might be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

RESIDENT
There is a single copy of the program that is not removed from storage unless it is deleted. RESIDENT programs must be at least quasi-reentrant. Any program of PROGRAM_TYPE SHARED has the RESIDENT attribute by default. The DELETE_PROGRAM call has no effect on this type of RESIDENT program.

REUSABLE
The program is similar to RESIDENT, except that if the program is not in use, CICS can remove it from storage to optimize storage use.

TRANSIENT
The program is similar to RESIDENT, except that the program is removed from storage as soon as its use count drops to zero.

PROGRAM_LENGTH(name4 | (Rn) | *)
Returns the length of the named program.

name4 The name of a 4-byte location to receive the length in bytes, expressed in binary.

(Rn) A register to contain the length in bytes, expressed in binary.

LOAD_POINT(name4 | (Ra) | *)
Returns the load point address of the program.

name4 The name of a 4-byte location to receive the loaded address.

(Ra) A register to contain the load address.

ENTRY_POINT(name4 | (Ra) | *)
Returns the entry point address of the program.

name4 The name of a 4-byte location to receive the 31-bit entry address.

(Ra) A register to receive the entry address.

RESPONSE and REASON values for IDENTIFY_PROGRAM

RESPONSE	REASON
OK	None
EXCEPTION	INSTANCE_NOT_FOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more information, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The RELEASE_PROGRAM call

RELEASE_PROGRAM decrements the use count of a currently loaded program by one.

If the program has been defined with the RELOAD attribute, the storage occupied by this copy of the program is released.

You should issue the ACQUIRE_PROGRAM and RELEASE_PROGRAM requests for a single program during the same execution of the exit program. If you do not want to do this, you should acquire the program once during CICS initialization, and leave it resident until CICS termination.

RELEASE_PROGRAM

```
DFHLDLDX [CALL,]
      [CLEAR,]
      [IN,
```

```
FUNCTION(RELEASE_PROGRAM),
ENTRY_POINT(pointer),
{PROGRAM_NAME(name8 | string | 'string')|
PROGRAM_TOKEN(name8)},]
[OUT,
RESPONSE(name1 | *),
REASON(name1 | *)]
```

This command is threadsafe.

ENTRY_POINT(pointer)
Specifies the address of the entry point of this copy of the named program.

PROGRAM_NAME(name8 | string | "string")
Specifies the name of the program to be released.

name8 The name of a location containing an 8-byte program name.

string A string of characters naming the program.

"string"
A string in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name8),
Specifies a token identifying the program to be released.

name8 The name of a location containing the 8-byte token obtained from a previous DEFINE_PROGRAM or ACQUIRE_PROGRAM call.

RESPONSE and REASON values for RELEASE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED
	PROGRAM_NOT_IN_USE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.
2. 'PROGRAM_NOT_DEFINED' is returned if the program that you name is not known to the system.
3. 'PROGRAM_NOT_IN_USE' is returned when the use count for the named program is already zero.

Log manager XPI functions

The XPI provides two log manager functions. These are the DFHLGPAX calls INQUIRE_PARAMETERS and SET_PARAMETERS. You can use these calls to inquire upon, and set, the log manager parameter, KEYPOINT_FREQUENCY. This parameter specifies the activity keypoint frequency of the CICS region.

The INQUIRE_PARAMETERS call

INQUIRE_PARAMETERS returns information about the activity keypoint frequency of the system.

INQUIRE_PARAMETERS

```
DFHLGPAX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_PARAMETERS),
      [OUT,
      [KEYPOINT_FREQUENCY(name4 | *),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

KEYPOINT_FREQUENCY(name4 | *)
returns the activity keypointing frequency of the CICS region.

name4 The name of a 4-byte location that is to receive the frequency value.

RESPONSE and REASON values for INQUIRE_PARAMETERS

RESPONSE	*REASON*
OK	None
DISASTER	None
INVALID	None
KERNERROR	None

The SET_PARAMETERS call

SET_PARAMETERS allows you to set the activity keypoint frequency for the CICS region.

SET_PARAMETERS

```
DFHLGPAX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SET_PARAMETERS),
      [KEYPOINT_FREQUENCY(name4 | (Rn) ),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

KEYPOINT_FREQUENCY(name4 | *)
specifies the activity keypointing frequency of the CICS region.

Permitted values are 0, or any integer between 200 and 65535 inclusive.

name4 The name of a 4-byte location that contains the new frequency value.

(Rn) A register that contains the new frequency value.

RESPONSE and REASON values for SET_PARAMETERS

RESPONSE	*REASON*
OK	None
EXCEPTION	OUT_OF_RANGE
DISASTER	None
INVALID	None
KERNERROR	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

Monitoring XPI functions

The XPI provides three monitoring functions: the DFHMNMNX calls INQUIRE_MONITORING_DATA and MONITOR, and the DFHMNIAX call INQUIRE_APP_CONTEXT.

Restriction:

DFHMNMNX and DFHMNIAX calls cannot be used in any exit program called from any global user exit point in the following domains:

- Dispatcher domain
- Dump domain
- Monitor domain
- Statistics domain
- Transient data program

In addition, INQUIRE_APP_CONTEXT cannot be used in any exit program called from any global user exit point in the following domains:

- Message domain
- Transaction manager domain

INQUIRE_APP_CONTEXT and INQUIRE_MONITORING_DATA calls cannot be used in any exit program called from any global user exit point in DFHTCP or DFHZCP.

The INQUIRE_APP_CONTEXT call

The INQUIRE_APP_CONTEXT call returns to the exit program the application context data that is associated with the current application running the issuing task.

Restriction

Do not start exit programs that use the INQUIRE_APP_CONTEXT function until the second phase of the PLTPI. For further information about the PLTPI, see Chapter 5, "Writing initialization and shutdown programs," on page 409.

INQUIRE_APP_CONTEXT

```
DFHMNIAX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_APP_CONTEXT),]
      [OUT,
      [APPLNAME(name64),]
      [PLATNAME(name64),]
      [OPERNAME(name64),]
      [MAJORVER(name4 | (Rn)),]
      [MINORVER(name4 | (Rn)),]
      [MICROVER(name4 | (Rn)),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

APPLNAME(name64)
Returns the 64-byte name of the current application associated with the task.

MAJORVER(name4 | (Rn))
Returns the major version number of the current application associated with the task.

name4 The name of a 4-byte field that contains the major version number as a binary value.

(Rn) A register to receive the major version number.

MICROVER(name4 | (Rn))
Returns the micro version number of the current application associated with the task.

name4 The name of a 4-byte field that contains the micro version number as a binary value.

(Rn) A register to receive the micro version number.

MINORVER(name4 | (Rn))
Returns the minor version number of the current application associated with the task.

name4 The name of a 4-byte field that contains the minor version number as a binary value.

(Rn) A register to receive the minor version number.

OPERNAME(name64)
Returns the 64-byte name of the operation associated with the task.

PLATNAME(name64)
Returns the 64-byte name of the platform associated with the task.

RESPONSE and REASON values for INQUIRE_APP_CONTEXT

RESPONSE	REASON
OK	None
EXCEPTION	APP_CONTEXT_UNAVAILABLE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

For more information, see the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The INQUIRE_MONITORING_DATA call

The INQUIRE_MONITORING_DATA function returns to the exit program the performance class monitoring data that has been accumulated for the issuing task.

The DFHMNTDS DSECT that maps the data is of fixed format. Note that:

- *All* the CICS system-defined fields in the performance records (including fields that you have specified for exclusion using the EXCLUDE option of the DFHMCT TYPE=RECORD macro) are listed.
- *No* user-defined data fields are listed.

INQUIRE_MONITORING_DATA

```
DFHMNMNX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_MONITORING_DATA),
      DATA_BUFFER(buffer-descriptor),]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Chapter 5, “Writing initialization and shutdown programs,” on page 409.

DATA_BUFFER(buffer-descriptor)
specifies the address and the length of a buffer to contain the returned monitoring data; see buffer-descriptor for a full definition of a buffer-descriptor. The DSECT DFHMNTDS maps the monitoring data.

RESPONSE and REASON values for INQUIRE_MONITORING_DATA

RESPONSE	*REASON*
OK	None
EXCEPTION	LENGTH_ERROR MONITOR_DATA_UNAVAILABLE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in “Making an XPI call” on page 382.
2. ‘LENGTH_ERROR’ means that the length specified in the buffer-descriptor was too short for the monitoring data returned from the XPI call.

The MONITOR call

The MONITOR XPI call is similar to the **EXEC CICS MONITOR** command. It enables you to invoke user event-monitoring points (EMPs) in your exit programs.

The user event-monitoring points must be defined in the monitoring control table (MCT) using the DFHMCT TYPE=EMP macro, or generated by the **APPLNAME** parameter on the DFHMCT TYPE=INITIAL macro. For more information about CICS monitoring, see CICS monitoring facility: Performance and tuning in Improving performance.

At a user EMP, you can add your own data (up to 256 counters, up to 256 clocks, and a single character string of up to 256 bytes) to fields reserved unconditionally

for you in performance class monitoring data records. You can also add up to 12 bytes of user information at the DFHAPPL EMPs.

MONITOR

```
DFHMNMNX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(MONITOR),
      POINT(expression | name2 | (Rn)),
      [DATA1(expression | name4 | (Ra) | *),]
      [DATA2(expression | name4 | (Ra) | *),]
      [ENTRYNAME(name8 | string | 'string'),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, see Chapter 5, “Writing initialization and shutdown programs,” on page 409.

DATA1(expression | name4 | (Ra) | *)
Specifies a fullword binary variable whose contents depend on the type of user EMP being used:

- If the MCT user EMP definition contains an ADDCNT, SUBCNT, NACNT, EXCNT, or ORCNT option, the DATA1 variable is an area used as defined by the user EMP definition.
- If the MCT user EMP definition contains an MLTCNT option, the DATA1 variable is an area with the address of a series of adjacent fullwords containing the values to be added to the user count fields defined in the user EMP definition.
- If the MCT user EMP definition contains a MOVE option, the DATA1 variable is an area with the address of the character string to be moved. This rule also applies to the DFHAPPL EMPs.

For details of the user EMP options, see Monitoring control table (MCT) in Reference -> System definition.

expression
A valid assembler-language expression giving the fullword binary variable for this EMP.

name4 The name of a 4-byte field containing the fullword binary variable for this EMP.

(Ra) A register containing the fullword binary variable for this EMP.

***** The value of this option is already present in the parameter list, or the option is not specified for this EMP.

DATA2(expression | name4 | (Rn) | *)
Specifies a fullword binary variable whose contents depend on the type of user EMP being used:

- If the MCT user EMP definition contains an ADDCNT, SUBCNT, NACNT, EXCNT, or ORCNT option, the DATA2 variable is an area used as defined by the user EMP definition.
- If the MCT user EMP definition contains an MLTCNT option, the DATA2 variable is an area with the number of user count fields to be updated.

  The number specified in DATA2 overrides the default value defined in the MCT for the operation. A value of 0 instructs monitoring to use the default. Not specifying a value for DATA2 does not prevent the MLTCNT operation from being successful; but, if it is, an exception response of DATA2_NOT_SPECIFIED is returned. See note 5.
- If the MCT user EMP definition contains a MOVE option, the DATA2 variable is an area with the length of the character string to be moved.

  The length specified in DATA2 overrides the default value defined in the MCT for the operation. A value of 0 instructs monitoring to use the default. Not specifying a value for DATA2 does not prevent the MOVE operation from being successful; but, if it is, an exception response of DATA2_NOT_SPECIFIED is returned. See note 5.

For details of the user EMP options, see Monitoring control table (MCT) in Reference -> System definition.

expression
A valid assembler-language expression giving the fullword binary variable for this EMP.

name4 The name of a 4-byte field containing the fullword binary variable for this EMP.

(Rn) A register containing the fullword binary variable for this EMP.

* The value of this option is already present in the parameter list, or the option is not specified for this EMP.

ENTRYNAME(name8 | string | "string")
Specifies the monitoring point entry name, which qualifies the POINT value and which is defined in the monitoring control table (MCT).

name8 The name of a location containing an 8-byte string.

string A string of characters without intervening blanks. The macro generates, from the string, a literal constant of length 8 bytes, extending with blanks or truncating as required.

"string"
A string, enclosed in quotation marks, and possibly containing blanks. This value is processed in the same way as the previous “string”.

Note: If, when defining the EMP in the MCT, you do not specify an entry name, the entry name defaults to ‘USER'. ENTRYNAME likewise defaults to ‘USER' if not specified.

POINT(expression | name2 | (Rn))
Specifies the monitoring point identifier as defined in the MCT, and is in the range 0 through 255. Point identifiers in the range 200 through 255 are reserved for use by IBM program products.

expression
A valid assembler-language expression that can be expressed in 2 bytes.

name2 The name of a 2-byte source of point data

(Rn) A register containing the point data in the low-order 2 bytes

RESPONSE and REASON values for MONITOR

RESPONSE	REASON
OK	None
EXCEPTION	DATA1_NOT_SPECIFIED
	DATA2_NOT_SPECIFIED
	POINT_NOT_DEFINED
	INVALID_DATA1_VALUE
	INVALID_DATA2_VALUE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, see the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.
2. POINT_NOT_DEFINED means that the EMP you have specified was not defined in the MCT.
3. INVALID_DATA1_VALUE and INVALID_DATA2_VALUE are most likely to have been caused by provision of bad addresses; this causes a program check.
4. DATA1_NOT_SPECIFIED and DATA2_NOT_SPECIFIED mean that you have not specified DATA1 or DATA2 respectively when the operation required them. See the description of DATA2.
5. Any error response terminates processing of the EMP. Operations defined to execute before the point of failure will have done so; later operations are canceled.

Object transaction XPI functions

You can use the object transaction XPI calls to implement a TRUE program that responds to calls between a CICS unit of work and a remote transaction coordinator. These are the DFHOTTRX calls COMMIT, COMMIT_ONE_PHASE, IMPORT_TRAN, PREPARE, ROLLBACK, and SET_ROLLBACK_ONLY, and the DFHOTCOX call SET_COORDINATOR. These functions provide powerful control over the sync point processing of a CICS unit of work. If you use these calls incorrectly, the CICS Recovery Manager terminates CICS immediately and recovery and resynchronization processing is required.

The IMPORT_TRAN call

Links the current unit of work of a task to an external transaction. Some information about the external transaction is recorded in the current unit of work.

IMPORT_TRAN

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(IMPORT_TRAN),
       [FORMAT_ID,(name4|Rn),]
       [BQUAL_LEN,(name4|Rn),]
       [TID_BLOCK_IN,(block-descriptor),]
       [TIMEOUT,(name4|Rn),]
```

```
[LOGICAL_SERVER,(name4|string|'string'),]
[PUBLIC_ID,(name64|string|'string'),]]
[OUT,
UOW_ID,(name8 | *),
RESPONSE (name1 | *),
REASON (name1 | *)]
```

This command is threadsafe.

BQUAL_LEN (name4 | Rn)
Specifies the branch qualifier length of the OTS transaction identifier (TID).
name4 The name of a 4-byte location containing the branch qualifier length.
Rn A register containing the branch qualifier length.

FORMAT_ID (name4 | Rn)
Specifies the OTS transactions format identifier.
name4 The name of a 4-byte location containing the format ID.
Rn A register containing the format ID.

LOGICAL_SERVER (name4 | string | 'string')
Specifies the name of the logical server in which the transaction is executing. Users of this XPI should choose values that are not similar to any CorbaServer definition that is installed in the CICS system.

PUBLIC_ID (name64 | string | 'string')
Specifies the public identifier associated with the transaction.

TID_BLOCK_IN (block-descriptor)
Specifies the unique OTS transaction identifier (TID) of the external transaction that will be associated with the task's unit of work. The block-descriptor is two fullwords of data, where the first word contains the address of the data, and the second word contains the length in bytes of the data.

TIMEOUT (name4 | Rn)
Specifies the OTS transaction timeout value in seconds.
name4 The name of a 4-byte location containing the timeout value.
Rn A register containing the timeout value.

UOW_ID (name8 | *)
Specifies the identifier of the CICS unit of work into which the OTS transaction was imported.

RESPONSE and REASON values for IMPORT_TRAN

RESPONSE	REASON
OK	None
EXCEPTION	TID_TOO_LONG OTS_TRAN_ALREADY
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The COMMIT_ONE_PHASE call

Performs a sync point on the unit of work of the current task, without referencing an external coordinator. You cannot use the COMMIT_ONE_PHASE call if you have used the SET_COORDINATOR call to add information about a coordinator to the unit of work of the current task.

COMMIT_ONE_PHASE

```
DFHOTTRX  [CALL,]
      [CLEAR,]
      [IN,
```

```
       FUNCTION(COMMIT_ONE_PHASE),]]
       [OUT,
       STATUS (name1 | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

STATUS (name1 | *)

The outcome of the CICS unit of work.

This parameter can have the following values:

COMMITTED
ROLLEDBACK

RESPONSE and REASON values for COMMIT_ONE_PHASE

RESPONSE	REASON
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The PREPARE call

Performs the first phase of the sync point on the CICS unit of work on behalf of an OTS transaction. The vote returned by this function is intended to be used by the coordinator of the OTS transaction to determine the outcome of the overall transaction. The CICS unit of work enters the *indoubt* state when PREPARE is called and remains indoubt until a subsequent COMMIT or ROLLBACK function is called. If the CICS system fails and needs to be restarted, the CICS unit of work is recovered from the system log and you must resynchronize to resolve the indoubt unit of work.

PREPARE

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(PREPARE),]
       [OUT,
       VOTE (name1 | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

VOTE (name1 | *)

The outcome of the first phase of the OTS transaction.

This parameter can have the following values:

YES
NO
READ_ONLY
HEURISTIC_MIXED

RESPONSE and REASON values for PREPARE

RESPONSE	REASON
OK	None

RESPONSE	REASON
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The COMMIT call

Performs the second phase of the sync point of an OTS transaction, ensuring that the transaction is committed.

COMMIT

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(COMMIT),]
       [OUT,
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

RESPONSE and REASON values for COMMIT

RESPONSE	REASON
OK	None
EXCEPTION	UOW_ROLLEDBACK
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The ROLLBACK call

Rolls back an OTS transaction.

ROLLBACK

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(ROLLBACK),]
       [OUT,
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

RESPONSE and REASON values for ROLLBACK

RESPONSE	REASON
OK	None
EXCEPTION	UOW_COMMITTED
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The SET_ROLLBACK_ONLY call

Marks the CICS unit of work so that the OTS coordinator is given a 'NO' vote when a sync point protocol is performed, forcing the rollback of the global transaction. The CICS unit of work continues in the *inflight* state, but any subsequent recoverable resource updates are rolled back with the rest of the global transaction.

SET_ROLLBACK_ONLY

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(SET_ROLLBACK_ONLY),]
       [OUT,
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

RESPONSE and REASON values for SET_ROLLBACK_ONLY

RESPONSE	REASON
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The SET_COORDINATOR call

Associates a link with the current task's unit of work to represent a remote coordinator.

SET_COORDINATOR

Ensure that the object transaction XPI functions for phase 1 (OTTR_PREPARE) and phase 2 (OTTR_COMMIT or OTTR_ROLLBACK) are used to drive the current unit of work through sync point processing, in response to the actions of the remote coordinator. You cannot use the OTTR_COMMIT_ONE_PHASE function if you have used the SET_COORDINATOR function to add information about a coordinator to the unit of work of the current task, and you cannot allow the task to end without using the correct object transaction XPIs to sync point the current unit of work.

```
DFHOTCOX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(SET_COORDINATOR),
       [IOR_BLOCK,(block-descriptor)]
       [HOST_BLOCK,(block-descriptor)]]
       [OUT,
       COORDINATOR_TOKEN,(name1 | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

HOST_BLOCK (block-descriptor)
A pointer to and length of a character string that represents the identity of the system, which contains the coordinator instance. The maximum supported length of this parameter is 4096 bytes.

IOR_BLOCK (block-descriptor)
A pointer to and length of a character string that represents the coordinator instance in the host system. The maximum supported length of this parameter is 4096 bytes.

COORDINATOR_TOKEN (name1 | *)
A token representing the coordinator.

RESPONSE and REASON values for SET_COORDINATOR

RESPONSE	REASON
OK	None
EXCEPTION	IOR_TOO LONG HOST_TOO_LONG LINK_UNKNOWN COORDINATOR_NOT_FOUND COORDINATOR_ALREADY INVALID_SYNCPOINT_STATE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Program management XPI functions

The XPI provides eight program management functions; six DFHPGISX calls and two DFHPGAQX calls. Together with the loader functions, these functions provide a comprehensive set of tools to manipulate programs.

The program management functions include the following DFHPGISX calls:

- END_BROWSE_PROGRAM
- GET_NEXT_PROGRAM
- INQUIRE_CURRENT_PROGRAM
- INQUIRE_PROGRAM
- SET_PROGRAM
- START_BROWSE_PROGRAM

The program management functions include the following DFHPGAQX calls:

- INQUIRE_AUTOINSTALL
- SET_AUTOINSTALL

You can use these functions together with the loader functions (DFHLDLDX calls) to manipulate programs. However, the tokens returned in the NEW_PROGRAM_TOKEN fields of DFHPGISX calls are different from the tokens that are returned by DFHLDLDX calls. Do not use a token obtained from a DFHPGISX call in a DFHLDLDX call, or vice versa.

The INQUIRE_PROGRAM call

INQUIRE_PROGRAM returns information about the attributes of a specified program.

INQUIRE_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_PROGRAM),
      [AC_APPLICATION_NAME(block-descriptor),]
      [AC_MAJOR_VERSION(name4),]
      [AC_MICRO_VERSION(name4),]
      [AC_MINOR_VERSION(name4),]
      [AC_PLATFORM_NAME(block-descriptor),]
      [{PROGRAM_NAME(name8 | string | 'string')|
      PROGRAM_TOKEN(name4)},]
      [SHOW_PROGRAMS(PRIVATE|PRIVATE_AND_PUBLIC),]
      [OUT,
      [ACCESS(CICS|NONE|READ_ONLY|USER),]
      [APIST(CICSAPI|OPENAPI),]
      [AVAIL_STATUS(DISABLED|ENABLED),]
      [CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC),]
      [CONCURRENCY(QUASIRENT|THREADSAFE),]
      [DATA_LOCATION(ANY|BELOW|NOT_APPLIC),]
      [DYNAMIC_STATUS(DYNAMIC|NOT_DYNAMIC),]
      [ENTRY_POINT(name4),]
      [EXECUTION_KEY(CICS|NOT_APPLIC|USER),]
      [EXECUTION_SET(DPLSUBSET|FULLAPI|NOT_APPLIC),]
      [HOLD_STATUS(CICS_LIFE|NOT_APPLIC|TASK_LIFE),]
      [INSTALL_TYPE(AUTO|CATALOG|GROUPLIST|MANUAL|RDO|SYSAUTO),]
      [LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|
                        COBOL2|LE370|NOT_APPLIC|NOT_DEDUCED|PLI),]
      [LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|
                        LE370|NOT_APPLIC|NOT_DEFINED|PLI),]
      [LIBRARY(name8),]
      [LIBRARYDSN(name44),]
      [LOAD_POINT(name4),]
      [LOAD_STATUS(LOADABLE|NOT_APPLIC|NOT_LOADABLE|NOT_LOADED),]
      [LOCATION(CDSA|ECDSA|ELPA|ERDSA|ESDSA|LPA|NONE|RDSA|SDSA),]
      [MODULE_TYPE(MAPSET|PARTITIONSET|PROGRAM),]
      [NEW_PROGRAM_TOKEN(name4),]
      [PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT),]
      [PROGRAM_LENGTH(name4),]
      [PROGRAM_TYPE(NOT_APPLIC|PRIVATE|SHARED|TYPE_ANY),]
      [PROGRAM_USAGE(APPLICATION|NUCLEUS),]
      [PROGRAM_USE_COUNT(name4),]
      [PROGRAM_USER_COUNT(name4),]
      [REMOTE_DEFINITION(LOCAL|REMOTE),]
      [REMOTE_PROGID(name8),]
      [REMOTE_SYSID(name4),]
      [REMOTE_TRANID(name4),]
      [SPECIFIED_AMODE(24|31|AMODE_ANY|AMODE_NOT_SPECIFIED|64),]
      [SPECIFIED_RMODE(24|RMODE_ANY|RMODE_NOT_SPECIFIED),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

AC_APPLICATION_NAME(block-descriptor)
Specifies the address and length of the name of the application associated with the program. To inquire on private programs for applications deployed on platforms, you must specify the AC_APPLICATION_NAME, AC_MAJOR_VERSION, AC_MINOR_VERSION, AC_MICRO_VERSION, and AC_PLATFORM_NAME fields to provide a complete application context. For more information on block-descriptors, see "XPI syntax" on page 393.

AC_MAJOR_VERSION(name4)
Specifies the application major version in binary.

AC_MICRO_VERSION(name4)
Specifies the application micro version in binary.

AC_MINOR_VERSION(name4)
Specifies the application minor version in binary.

AC_PLATFORM_NAME(block-descriptor)
Specifies the address and length of the name of the platform associated with the program. For more information on block-descriptors, see "XPI syntax" on page 393.

ACCESS(CICS|NONE|READ_ONLY|USER)
Returns a value that indicates the type of storage into which the program has been loaded.

CICS CICS-key.

NONE
The program has not been loaded.

READ_ONLY
Read-only.

USER User-key.

APIST(CICSAPI|OPENAPI)
Returns a value that indicates the API attribute of the installed program definition.

CICSAPI
The program is restricted to use of only the CICS permitted application programming interfaces.

OPENAPI
The program is not restricted to use of only the CICS permitted application programming interfaces. The program must be coded to threadsafe standards and defined with CONCURRENCY(THREADSAFE).

AVAIL_STATUS(DISABLED|ENABLED)
Returns a value that indicates whether the program can be used (ENABLED) or not (DISABLED).

CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC)
Returns the execution diagnostic facility (EDF) status of the program.

CEDF When the program is running under the control of the CICS EDF, EDF diagnostic screens are displayed.

NOCEDF
EDF diagnostic screens are not displayed.

NOT_APPLIC
Not applicable. This module is a mapset, a partitionset, or a remote program.

CONCURRENCY(QUASIRENT|THREADSAFE)
Returns a value that indicates the concurrency attribute of the installed program definition.

QUASIRENT
The program is defined as being quasi-reentrant, and can run only under the CICS QR TCB.

THREADSAFE
The program is defined as threadsafe, and can run under whichever TCB is in use by its user task when the program is given control. This could be either an open TCB or the CICS QR TCB.

Note: For a Language Environment-conforming program, the concurrency as originally defined can be overridden when the program is subsequently loaded.

DATA_LOCATION(ANY|BELOW|NOT_APPLIC)
Returns a value that indicates whether the program can access data located above the 16 MB line.

ANY The program can handle 31-bit addresses, and can be passed data that is located above or below the 16 MB line.

BELOW
The program can handle only 24-bit addresses, and must therefore be passed only data that is located below the 16 MB line.

NOT_APPLIC
Not applicable. This module is a mapset, a partitionset, or a remote program.

DYNAMIC_STATUS(DYNAMIC|NOT_DYNAMIC)
Returns a value that indicates whether, if the program is the subject of a program-link request, the request can be dynamically routed.

DYNAMIC
If the program is the subject of a program-link request, the CICS dynamic routing program is invoked. Providing that a remote server region is not named explicitly on the SYSID option of the EXEC CICS LINK command, the routing program can route the request to the region on which the program is to run.

NOT_DYNAMIC
If the program is the subject of a program-link request, the dynamic routing program is not invoked.

For a distributed program link (DPL) request, the server region on which the program is to run must be specified explicitly on the REMOTESYSTEM option of the PROGRAM definition, or on the SYSID option of the EXEC CICS LINK command. Otherwise, it defaults to the local region.

For information about the dynamic routing of DPL requests, see the *CICS Intercommunication Guide*.

ENTRY_POINT(name4)
Returns the entry point address of the program entry point address, as returned by a loader domain ACQUIRE_PROGRAM call.

EXECUTION_KEY(CICS|NOT_APPLIC|USER)
Returns the key in which CICS gives control to the program, which determines whether the program can modify CICS-key storage.

CICS CICS gives control to the program in CICS key. The program is loaded into a CICS dynamic storage area (DSA) above or below the 16 MB line; that is, the CDSA or ECDSA, depending on its residency mode (RMODE) attribute, as defined to the linkage-editor.

NOT_APPLIC
Not applicable. This module is a mapset, a partitionset, or a remote program.

USER CICS gives control to the program in user key. The program is loaded into a user DSA above or below the 16 MB line; that is, the UDSA or EUDSA, depending on its residency mode (RMODE) attribute, as defined to the linkage-editor.

EXECUTION_SET(DPLSUBSET|FULLAPI|NOT_APPLIC)
Returns a value that indicates whether CICS links to and runs the program as if it were running in a remote CICS region.

DPLSUBSET
CICS links to and runs the program with the API restrictions of a remote DPL program. The program can use only a subset of the CICS API.

FULLAPI
CICS links to and runs the program without the API restrictions of a remote DPL program. The program can use the full CICS API.

NOT_APPLIC
Not applicable. This module is a mapset, a partitionset, or a remote program. (The EXECUTIONSET option of DEFINE PROGRAM applies only to local program definitions. Its purpose is to test programs in a local CICS environment as if they were running as DPL programs.)

HOLD_STATUS(CICS_LIFE|NOT_APPLIC|TASK_LIFE)
Returns a value that indicates how long the program is to remain loaded.

CICS_LIFE
The program remains loaded until CICS is shut down.

NOT_APPLIC
Not applicable. The program is not loaded, or is remote.

TASK_LIFE
The program remains loaded for the lifetime of the task.

INSTALL_TYPE(AUTO|CATALOG|GROUPLIST|MANUAL|RDO|SYSAUTO)
Returns the method that was used to install the PROGRAM resource definition.

AUTO
Autoinstall.

CATALOG
The CICS global catalog, after a restart.

GROUPLIST
The CICS startup grouplist.

MANUAL
The program is a CICS internal module explicitly defined to the Program Manager by another CICS component.

RDO RDO commands.

SYSAUTO
System autoinstall (that is, autoinstalled by CICS without calling the autoinstall user program). The program might be a CICS internal module or, for example, a first phase PLTPI program.

LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|COBOL2|LE370|NOT_APPLIC|NOT_DEDUCED|PLI)
Returns the language deduced by CICS for the program. COBOL is OS/VS COBOL, which cannot run under this CICS version, and COBOL2 is either Enterprise COBOL or VS COBOL II.

LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|LE370| NOT_APPLIC|NOT_DEFINED|PLI)
Returns the programming language specified on the resource definition.

LIBRARY(name)
Returns the 8-character name of the LIBRARY resource from which this program was loaded. This field is blank if the program has not been loaded, or if the LPASTATUS is LPA (indicating that the program has been loaded from the LPA).

LIBRARYDSN(name44)
Returns the 44-character name of the data set from which the program was loaded. This field is blank if the program has not been loaded, or if the LPASTATUS is LPA (indicating that the program has been loaded from the LPA).

- If the program was loaded from an installed LIBRARY, the LIBRARY and LIBRARYDSN names will be returned.
- If the program was loaded from a LIBRARY that has been disabled, the LIBRARY name will be returned but the LIBRARYDSN will be blank.
- If the program was loaded from a LIBRARY that has been discarded, both LIBRARY and LIBRARYDSN will be blank.

LOAD_POINT(name4)
Returns the load point address of the program, as returned by a loader domain ACQUIRE_PROGRAM call.

LOAD_STATUS(LOADABLE|NOT_APPLIC|NOT_LOADABLE|NOT_LOADED)
Returns a value that indicates whether or not the program can be loaded.

LOADABLE
The program is loadable.

NOT_APPLIC
Not applicable. The program is remote.

NOT_LOADABLE
CICS has tried to load the program and failed; the program is not in the library.

NOT_LOADED
CICS has not yet tried to load the program.

LOCATION(CDSA|ECDSA|ELPA|ERDSA|ESDSA|LPA|NONE|RDSA|SDSA)
Returns a value that indicates where the most recently loaded copy of the program resides.

CDSA The CICS dynamic storage area

ECDSA
The extended CICS dynamic storage area

ELPA The extended link pack area

ERDSA
The extended read-only dynamic storage area

ESDSA
The extended shared dynamic storage area

LPA The link pack area

NONE
The program has not been loaded.

RDSA The read-only dynamic storage area

SDSA The shared dynamic storage area

MODULE_TYPE(MAPSET|PARTITIONSET|PROGRAM)
Returns the type of program resource.

NEW_PROGRAM_TOKEN(name4)
Returns a token that can be used to identify the named program.

name4 The name of a location to receive a 4-byte token that identifies this program.

If PROGRAM_NAME is specified on the request, NEW_PROGRAM_TOKEN is set to a program token that can be used on subsequent requests for the same program. If PROGRAM_TOKEN is specified on the request, NEW_PROGRAM_TOKEN is set to the same value.

PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT)
Returns the residency status of the program; that is, when its storage is released.

RELOAD
The program is not reusable, and therefore several copies might be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

RESIDENT
There is a single copy of the program that is not removed from storage unless deleted. RESIDENT programs must be at least quasi-reentrant. Any program of PROGRAM_TYPE SHARED is RESIDENT by default.

REUSABLE
Similar to RESIDENT, except that CICS can remove a REUSABLE program that is not in use from storage, for storage optimization reasons.

TRANSIENT
Similar to RESIDENT, except that a TRANSIENT program is removed from storage as soon as its user count drops to zero.

PROGRAM_LENGTH(name4)
Returns the length of the program, in bytes, expressed in binary.

PROGRAM_NAME(name8 | string | 'string')
Specifies the name of the program to be queried.

name8 The name of a location that contains an 8-byte program name.

string A string of characters that name the program.

'string'
A string of characters in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name4)
Specifies a token that identifies the program to be queried.

name4 The name of a location that contains a 4-byte token that was obtained from a previous INQUIRE_PROGRAM call.

PROGRAM_TYPE(NOT_APPLIC|PRIVATE|SHARED|TYPE_ANY)
Returns a value that indicates where the next new copy of the program is to be loaded from.

NOT_APPLIC
Not applicable. The program is remote.

PRIVATE
The program is to be loaded from the DFHRPL or dynamic LIBRARY concatenation.. A PRIVATE program need not be reentrant, and is given only limited protection against unauthorized overwriting. The degree of protection depends on the type of dynamic storage area into which the program is loaded (see the description of the PROGRAM_TYPE option of the DEFINE_PROGRAM call).

SHARED
The program is to be loaded from the link pack area (LPA). SHARED programs must be reentrant, and are protected.

The next time a NEWCOPY or PHASEIN is received, an LPA copy of the program is used if it is available. If no LPA version is available, the program is loaded from DFHRPL or dynamic LIBRARY concatenation.

TYPE_ANY
Either the copy in DFHRPL or a dynamic LIBRARY concatenation, or the LPA copy of the program can be used, although preference is given to the LPA copy.

PROGRAM_USAGE(APPLICATION|NUCLEUS)
Returns a value that indicates whether the program is used as a CICS nucleus program, or as a user application program.

PROGRAM_USE_COUNT(name4)
Returns the number of different users that have invoked the program.

PROGRAM_USER_COUNT(name4)
Returns the current number of users of the program.

REMOTE_DEFINITION(LOCAL|REMOTE)
Returns a value that indicates whether this program is a local or a remote resource. If it is a remote resource, CICS treats requests to link to the program as distributed program link (DPL) requests, and ships them to the remote region.

REMOTE_PROGID(name8)
Returns the name by which the program is known in the remote CICS region, if the program is a remote resource. If REMOTESYSTEM was specified on the PROGRAM definition, and REMOTENAME omitted, the remote name will be the same as the local name (that is, REMOTE_PROGID will default to the value of PROGRAM_NAME).

REMOTE_SYSID(name4)
Returns the name of the remote CICS region that owns the program, if the program is a remote resource.

REMOTE_TRANID(name4)
Returns the name of the transaction that the remote CICS attaches, and under which it runs the program, if the program is a remote resource.

SHOW_PROGRAMS(PRIVATE|PRIVATE_AND_PUBLIC)
If application context fields are specified for INQUIRE_PROGRAM, SHOW_PROGRAMS defines the scope of the search.

PRIVATE
Searches only for private programs.

PRIVATE_AND_PUBLIC
Searches for private programs first, then public programs.

SPECIFIED_AMODE(24|31|AMODE_ANY|AMODE_NOT_SPECIFIED|64)
Returns the addressing mode that was specified on a DEFINE_PROGRAM call.

SPECIFIED_RMODE(24|RMODE_ANY|RMODE_NOT_SPECIFIED)
Returns the residency mode (that is, whether the program should be loaded above or below the 16 MB line) that was specified on a DEFINE_PROGRAM call.

RESPONSE and REASON values for INQUIRE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED_TO_LD
	PROGRAM_NOT_DEFINED_TO_PG
	APP_CONTEXT_NOT_FOUND
DISASTER	ABEND
	LOCK_ERROR
INVALID	INVALID_PROGRAM_TOKEN
KERNERROR	None
PURGED	None

The INQUIRE_CURRENT_PROGRAM call

INQUIRE_CURRENT_PROGRAM returns information about the attributes of the program that is currently running. If this call is issued from within a global or task-related user exit, it returns the attributes of the global or task-related user exit program itself.

INQUIRE_CURRENT_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_CURRENT_PROGRAM),]
      [IGNORE_EXITS(YES|NO),]
      [OUT,
      [AVAIL_STATUS(DISABLED|ENABLED),]
      [CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC),]
      [CURRENT_AMODE(24|31|64),]
      [CURRENT_CEDF_STATUS(CEDF|NOCEDF),]
      [CURRENT_ENTRY_POINT(name4),]
      [CURRENT_ENVIRONMENT(EXEC|GLUE|PLT|SYSTEM|TRUE|URM),]
      [CURRENT_EXECUTION_SET(DPLSUBSET|FULLAPI),]
      [CURRENT_LOAD_POINT(name4),]
      [CURRENT_PROGRAM_LENGTH(name4),]
      [CURRENT_PROGRAM_NAME(name8),]
      [DATA_LOCATION(ANY|BELOW|NOT_APPLIC),]
      [DYNAMIC_STATUS(DYNAMIC|NOT_DYNAMIC),]
      [EXECUTION_KEY(CICS|NOT_APPLIC|USER),]
      [EXECUTION_SET(DPLSUBSET|FULLAPI|NOT_APPLIC),]
      [HOLD_STATUS(CICS_LIFE|NOT_APPLIC|TASK_LIFE),]
      [IGNORE_EXITS(YES|NO),]
      [INSTALL_TYPE(AUTO|CATALOG|GROUPLIST|MANUAL|RDO|SYSAUTO),]
      [INVOKING_ENVIRONMENT (EXEC|GLUE|PLT|SYSTEM|TRUE|URM),]
      [INVOKING_PROGRAM_NAME(name8),]
      [LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|
                        COBOL2|LE370|NOT_APPLIC|NOT_DEDUCED|PLI),]
```

```
[LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|
                  LE370|NOT_APPLIC|NOT_DEFINED|PLI),]
[LIBRARY(name8),]
[LIBRARYDSN(name44),]
[LOAD_STATUS(LOADABLE|NOT_APPLIC|NOT_LOADABLE|NOT_LOADED),]
[MODULE_TYPE(MAPSET|PARTITIONSET|PROGRAM),]
[NEW_PROGRAM_TOKEN(name4),]
[REMOTE_DEFINITION(LOCAL|REMOTE),]
[REMOTE_PROGID(name8),]
[REMOTE_SYSID(name4),]
[REMOTE_TRANID(name4),]
[RETURN_PROGRAM_NAME(name8),]
RESPONSE(name1 | *),
REASON(name1 | *)]
```

This command is threadsafe.

Note: The options not described in the following list are identical to the equivalent options of the INQUIRE_PROGRAM call. See "The INQUIRE_PROGRAM call" on page 818.

CURRENT_AMODE(24|31|64)
Returns the addressing mode that the running program is currently using.

CURRENT_CEDF_STATUS(CEDF|NOCEDF)
Returns the EDF status of the current instance of the program. The value returned is the same as for CEDF_STATUS, which is the EDF status specified on the program definition. See the CEDF_STATUS option of INQUIRE_PROGRAM.

CURRENT_ENTRY_POINT(name4)
Returns the entry point address of the current program.

CURRENT_ENVIRONMENT(EXEC|GLUE|PLT|SYSTEM|TRUE|URM)
Returns the environment in which the current program is running; that is, the type of program it is.

EXEC User application program.

GLUE Global user exit program.

PLT Program list table program.

SYSTEM
CICS system code.

TRUE Task-related user exit program.

URM User-replaceable program.

CURRENT_EXECUTION_SET(DPLSUBSET|FULLAPI)
Returns the API execution set used by the current instance of the program. The value returned is the same as for EXECUTION_SET (which is the API execution set specified on the program definition) *unless* this is the first program in a transaction, when the value can be different. This is because the DPLSUBSET attribute applies only to linked-to programs. It is ignored for the first program in a transaction, because this cannot be the target of a DPL call. Therefore, for the first program in a transaction, if EXECUTION_SET returns DPLSUBSET, CURRENT_EXECUTION_SET nevertheless returns FULLAPI. See the EXECUTION_SET option of INQUIRE_PROGRAM.

CURRENT_LOAD_POINT(name4)
Returns the load point address of the current program.

CURRENT_PROGRAM_LENGTH(name4)
Returns the length of the current program, in bytes, expressed in binary.

CURRENT_PROGRAM_NAME(name8)
Returns the name of the program that is currently running.

IGNORE_EXITS(YES|NO)
Specifies whether global user exit programs and task-related user exit programs are ignored when returning information about the program that invoked the current program, and to which control will be returned. Your setting for this option affects the values returned by the INVOKING_ENVIRONMENT, INVOKING_PROGRAM_NAME, and RETURN_PROGRAM_NAME options. If YES is specified (the default), global user exit programs and task-related user exit programs are ignored for these options. If NO is specified, where a global user exit program or task-related user exit program is involved, the options return information about the exit program.

INVOKING_ENVIRONMENT (EXEC|GLUE|PLT|SYSTEM|TRUE|URM)
Returns the environment from which the current program was invoked; that is, the environment that corresponds to the program named in INVOKING_PROGRAM_NAME. The values are as described for CURRENT_ENVIRONMENT.

INVOKING_PROGRAM_NAME(name8)
Returns the name of the most recent program to invoke the current program. If IGNORE_EXITS(NO) is specified, this might be a global user exit program or task-related user exit program, if one was involved. If IGNORE_EXITS(YES) is specified, which is the default, this is the most recent program that was *not* a global user exit program or task-related user exit program.

LIBRARY(name)
Returns the 8-character name of the LIBRARY resource from which this program was loaded. This field is blank if the program has not been loaded, or if the LPASTATUS is LPA (indicating that the program has been loaded from the LPA). If the program was loaded from an installed LIBRARY, the LIBRARY and LIBRARYDSN names are returned.

LIBRARYDSN(*data-area*)
Returns the 44-character name of the data set from which the program was loaded. This fields is blank if the program has not been loaded, or if the LPASTATUS is LPA (indicating that the program has been loaded from the LPA). If the program was loaded from an installed LIBRARY, the LIBRARY and LIBRARYDSN names are returned.

RETURN_PROGRAM_NAME(name8)
Returns the name of the program to which control will be returned. If IGNORE_EXITS(NO) is specified, this might be a global user exit program or task-related user exit program. If IGNORE_EXITS(YES) is specified, which is the default, this is the program to which control will be returned after any intermediate global user exit programs or task-related user exit programs have completed.

RESPONSE and REASON values for INQUIRE_CURRENT_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	NO_CURRENT_PROGRAM

RESPONSE	*REASON*
DISASTER	LOCK_ERROR
	ABEND
INVALID	None
KERNERROR	None
PURGED	None

The SET_PROGRAM call

Use SET_PROGRAM to set selected attributes in the definition of a specified program.

SET_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SET_PROGRAM),
      {PROGRAM_NAME(name8 | string | 'string')|
      PROGRAM_TOKEN(name4)},]
      [AVAIL_STATUS(DISABLED|ENABLED),]
      [CEDF_STATUS(CEDF|NOCEDF),]
      [EXECUTION_KEY(CICS|USER),]
      [EXECUTION_SET(DPLSUBSET|FULLAPI),]
      [PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT),]
      [PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY),]
      [PROGRAM_USAGE(APPLICATION|NUCLEUS),]
      [REQUIRED_AMODE(24|31|AMODE_ANY|64),]
      [REQUIRED_RMODE(24|RMODE_ANY),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

AVAIL_STATUS(DISABLED|ENABLED)
Specifies whether the program can be used (ENABLED) or not (DISABLED).

CEDF_STATUS(CEDF|NOCEDF)
Specifies whether, when the program is running under the control of the CICS execution diagnostic facility (EDF), EDF diagnostic screens are displayed.

EXECUTION_KEY(CICS|USER)
Specifies the key in which CICS is to give control to the program. The key determines whether the program can modify CICS-key storage.

CICS CICS gives control to the program in CICS key. The program is loaded into a CICS dynamic storage area (DSA) above or below the 16 MB line; that is, the CDSA or ECDSA, depending on its residency mode (RMODE) attribute, as defined to the linkage-editor.

USER CICS gives control to the program in user key. The program is loaded into a user DSA above or below the 16 MB line; that is, the UDSA or EUDSA, depending on its residency mode (RMODE) attribute, as defined to the linkage-editor.

Note: If the program has been link-edited as reentrant with AMODE(31),RMODE(ANY), the EXECUTION_KEY option is ignored, and it is loaded into the extended read-only DSA (ERDSA). For details of the type of storage allocated for the ERDSA, see RENTPGM system initialization parameter in Reference -> System definition.

EXECUTION_SET(DPLSUBSET|FULLAPI)
Specifies whether CICS is to link to and run the program as if it were running in a remote CICS region.

Note: EXECUTION_SET applies only to local program definitions. Its purpose is to test programs in a local CICS environment as if they were running as DPL programs.

DPLSUBSET
CICS links to and runs the program with the API restrictions of a remote DPL program. The program can use only a subset of the CICS API.

FULLAPI
CICS links to and runs the program without the API restrictions of a remote DPL program. The program can use the full CICS API.

PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT)
Specifies the residency status of the program; that is, when its storage is to be released.

RELOAD
The program is not reusable, and therefore several copies might be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

RESIDENT
At any one time there will be no more than a single copy of the program in storage, and this will not be removed unless deleted. RESIDENT programs must be at least quasi-reentrant. Any program of PROGRAM_TYPE SHARED is RESIDENT by default.

REUSABLE
Similar to RESIDENT, except that CICS can remove a REUSABLE program that is not in use from storage, for storage optimization reasons.

TRANSIENT
Similar to RESIDENT, except that a TRANSIENT program is removed from storage as soon as its user count drops to zero.

PROGRAM_NAME(name8 | string | 'string')
Specifies the name of the program whose attributes are to be changed.

name8 The name of a location that contains an 8-byte program name.

string A string of characters that name the program.

'string'
A string of characters in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name4)
Specifies a token that identifies the program.

name4 The name of a location that contains a 4-byte token that was obtained from a previous INQUIRE_PROGRAM, INQUIRE_CURRENT_PROGRAM, START_BROWSE_PROGRAM, or GET_NEXT_PROGRAM call.

PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY)
Specifies where the program is to be loaded from.

PRIVATE
The program is in the DFHRPL or dynamic LIBRARY concatenation. A PRIVATE program need not be reentrant, and is given only limited protection against unauthorized overwriting. The degree of protection depends on the type of dynamic storage area into which the program is loaded (see the description of the PROGRAM_TYPE option of the DEFINE_PROGRAM call).

SHARED
The program is located in the link pack area (LPA), is reentrant, and is protected.

TYPE_ANY
Either the copy in the DFHRPL or a dynamic LIBRARY concatenation, or the LPA copy of the program can be used, though preference is given to the LPA copy.

PROGRAM_USAGE(APPLICATION|NUCLEUS)
Specifies whether the program is used as a CICS nucleus program, or as a user application program.

REQUIRED_AMODE(24|31|AMODE_ANY|64)
Specifies the addressing mode of the program. If, during subsequent processing, no copy of the program that meets the defined addressing requirement can be found, an exception occurs.

Note:

1. AMODE_ANY and 31 have identical meanings for this function.
2. You cannot use this option to override the link-edited addressing mode of the program.

REQUIRED_RMODE(24|AMODE_ANY)
Specifies the residency mode of the program (that is, whether it is to be loaded above or below the 16MB line). If, during subsequent processing, no copy of the program that meets the defined residency requirement can be found, an exception occurs.

Note: You cannot use this option to override the link-edited residency mode of the program.

RESPONSE and REASON values for SET_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	CEDF_STATUS_NOT_FOR_MAPSET
	CEDF_STATUS_NOT_FOR_PTNSET
	CEDF_STATUS_NOT_FOR_REMOTE
	EXEC_KEY_NOT_FOR_MAPSET
	EXEC_KEY_NOT_FOR_PTNSET
	EXEC_KEY_NOT_FOR_REMOTE
	EXEC_SET_NOT_FOR_MAPSET
	EXEC_SET_NOT_FOR_PTNSET
	EXEC_SET_NOT_FOR_REMOTE
	INCOMPATIBLE_BUNDLE_SET
	PROGRAM_NOT_DEFINED_TO_LD
	PROGRAM_NOT_DEFINED_TO_PG
DISASTER	ABEND
	CATALOG_ERROR

RESPONSE	*REASON*
	CATALOG_NOT_OPERATIONAL
	LOCK_ERROR
INVALID	INVALID_MODE_COMBINATION
	INVALID_PROGRAM_NAME
	INVALID_PROGRAM_TOKEN
	INVALID_TYPE_ATTRIB_COMBIN
KERNERROR	None
PURGED	None

The START_BROWSE_PROGRAM call

START_BROWSE_PROGRAM returns a token that enables you to begin browsing through program definitions, optionally starting at the definition of a specified program.

START_BROWSE_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(START_BROWSE_PROGRAM),
      [PROGRAM_NAME(name8 | string | 'string'),]
      [OUT,
      BROWSE_TOKEN(name4)
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

BROWSE_TOKEN(name4)
returns a token to be used on a GET_NEXT_PROGRAM call, to initiate a sequential browse of program definitions.

name4 The name of a location to receive a 4-byte token.

PROGRAM_NAME(name8 | string | 'string')
specifies the name of the program whose definition you want to look at first. The browsing sequence is alphabetical. If there is no program with the specified name, CICS returns a token for the next definition in the alphabetic sequence. If you do not specify a program, CICS returns a token for the first definition.

name8 The name of a location containing an 8-byte program name.

string A string of characters naming the program.

'string'
A string of characters in quotation marks. The string length is set to 8 by padding with blanks or truncating.

RESPONSE and REASON values for START_BROWSE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	
DISASTER	ABEND
	INVALID_DIRECTORY
	LOCK_ERROR

RESPONSE	*REASON*
INVALID	None
KERNERROR	None
PURGED	None

The GET_NEXT_PROGRAM call

Use GET_NEXT_PROGRAM to inquire on the next program definition during a browse sequence that is initiated by START_BROWSE_PROGRAM. The browsing sequence is alphabetical. The end of the alphabetical list of definitions is indicated by an END_LIST exception response.

GET_NEXT_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(GET_NEXT_PROGRAM),
      BROWSE_TOKEN(name4),]
      [OUT,
      PROGRAM_NAME(name8),
      [ACCESS(CICS|NONE|READ_ONLY|USER),]
      [AVAIL_STATUS(DISABLED|ENABLED),]
      [CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC),]
      [DATA_LOCATION(ANY|BELOW|NOT_APPLIC),]
      [ENTRY_POINT(name4),]
      [EXECUTION_KEY(CICS|NOT_APPLIC|USER),]
      [EXECUTION_SET(DPLSUBSET|FULLAPI|NOT_APPLIC),]
      [HOLD_STATUS(CICS_LIFE|NOT_APPLIC|TASK_LIFE),]
      [INSTALL_TYPE(AUTO|CATALOG|GROUPLIST|MANUAL|RDO|SYSAUTO),]
      [LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|
                        COBOL2|LE370|NOT_APPLIC|NOT_DEDUCED|PLI),]
      [LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|
                        LE370|NOT_APPLIC|NOT_DEFINED|PLI),]
      [LOAD_POINT(name4),]
      [LOAD_STATUS(LOADABLE|NOT_APPLIC|NOT_LOADABLE|NOT_LOADED),]
      [LOCATION(CDSA|ECDSA|ELPA|ERDSA|ESDSA|LPA|NONE|RDSA|SDSA),]
      [MODULE_TYPE(MAPSET|PARTITIONSET|PROGRAM),]
      [NEW_PROGRAM_TOKEN(name4),]
      [PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT),]
      [PROGRAM_LENGTH(name4),]
      [PROGRAM_TYPE(NOT_APPLIC|PRIVATE|SHARED|TYPE_ANY),]
      [PROGRAM_USAGE(APPLICATION|NUCLEUS),]
      [PROGRAM_USE_COUNT(name4),]
      [PROGRAM_USER_COUNT(name4),]
      [REMOTE_DEFINITION(LOCAL|REMOTE),]
      [REMOTE_PROGID(name8),]
      [REMOTE_SYSID(name4),]
      [REMOTE_TRANID(name4),]
      [SPECIFIED_AMODE(24|31|AMODE_ANY|AMODE_NOT_SPECIFIED|64),]
      [SPECIFIED_RMODE(24|RMODE_ANY|RMODE_NOT_SPECIFIED),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

Note: The options not described in the following list are identical to the equivalent options of the INQUIRE_PROGRAM call. See "The INQUIRE_PROGRAM call" on page 818.

BROWSE_TOKEN(name4)
Specifies a token that identifies the definition to be browsed. This can be either the token returned in the NEW_PROGRAM_TOKEN field of the last

GET_NEXT_PROGRAM call, or that in the BROWSE_TOKEN field of the START_BROWSE_PROGRAM call (this token is updated after every GET_PROGRAM call).

name4 The name of a location that contains a 4-byte token.

NEW_PROGRAM_TOKEN(name4)
Returns a token that identifies the next definition in the browse sequence. You can use it in the BROWSE_TOKEN field of your next GET_NEXT_PROGRAM call (or END_BROWSE_PROGRAM call, if you want to end the sequence). You can also use it in the PROGRAM_TOKEN field of INQUIRE_PROGRAM and SET_PROGRAM calls.

name4 The name of a location to receive a 4-byte token that identifies the next program definition.

RESPONSE and REASON values for GET_NEXT_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	END_LIST
	INVALID_BROWSE_TOKEN
	PROGRAM_NOT_DEFINED_TO_LD
DISASTER	ABEND
	LOCK_ERROR
INVALID	None
KERNERROR	None
PURGED	None

The END_BROWSE_PROGRAM call

END_BROWSE_PROGRAM allows you to end a browse of program definitions initiated by START_BROWSE_PROGRAM.

END_BROWSE_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(END_BROWSE_PROGRAM),
      BROWSE_TOKEN(name4),]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

BROWSE_TOKEN(name4)
specifies either the token returned in the NEW_PROGRAM_TOKEN field of the last GET_NEXT_PROGRAM call, or that in the BROWSE_TOKEN field of the START_BROWSE_PROGRAM call (this token is updated after every GET_NEXT_PROGRAM call).

RESPONSE and REASON values for END_BROWSE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_BROWSE_TOKEN
DISASTER	ABEND
	LOCK_ERROR

RESPONSE	*REASON*
INVALID	None
KERNERROR	None
PURGED	None

The INQUIRE_AUTOINSTALL call

INQUIRE_AUTOINSTALL returns information about the current settings of the autoinstall function for programs, mapsets, and partitionsets.

INQUIRE_AUTOINSTALL

```
DFHPGAQX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_AUTOINSTALL),]
      [OUT,
      [AUTOINSTALL_CATALOG (ALL|MODIFY|NONE),]
      [AUTOINSTALL_EXIT_NAME(name8),]
      [AUTOINSTALL_STATE (ACTIVE|INACTIVE),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

AUTOINSTALL_CATALOG(ALL|MODIFY|NONE)
returns the catalog status for autoinstalled program definitions.

ALL All autoinstalled program, map, and partitionset definitions are cataloged.

MODIFY
Autoinstalled program, map, and partitionset definitions are recorded on the CICS global catalog only if they are modified by a SET PROGRAM command after being autoinstalled.

NONE
No autoinstalled program, map, or partitionset definitions are cataloged.

AUTOINSTALL_EXIT_NAME(name8)
returns the name of the user-replaceable autoinstall control program for programs, mapsets, and partitionsets.

AUTOINSTALL_STATE(ACTIVE|INACTIVE)
returns the status of the program autoinstall function.

ACTIVE
Autoinstall is enabled for programs, mapsets, and partitionsets.

INACTIVE
Autoinstall is not enabled for programs, mapsets, and partitionsets.

RESPONSE and REASON values for INQUIRE_AUTOINSTALL

RESPONSE	*REASON*
OK	None
EXCEPTION	None
DISASTER	None
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The SET_AUTOINSTALL call

SET_AUTOINSTALL enables you to change the settings of the autoinstall function for programs, mapsets, and partitionsets.

SET_AUTOINSTALL

```
DFHPGAQX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SET_AUTOINSTALL),
      [AUTOINSTALL_CATALOG (ALL|MODIFY|NONE),]
      [AUTOINSTALL_EXIT_NAME(name8),]
      [AUTOINSTALL_STATE (ACTIVE|INACTIVE),]
      [LANGUAGES_AVAILABLE(NO|YES),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

AUTOINSTALL_CATALOG(ALL|MODIFY|NONE)
specifies the catalog status for autoinstalled program definitions.

ALL All autoinstalled program, map, and partitionset definitions are to be cataloged.

MODIFY
Autoinstalled program, map, and partitionset definitions are to be recorded on the CICS global catalog only if they are modified by a SET PROGRAM command after being autoinstalled.

NONE
No autoinstalled program, map, or partitionset definitions are to be cataloged.

AUTOINSTALL_EXIT_NAME(name8)
specifies the name of the user-replaceable autoinstall control program for programs, mapsets, and partitionsets.

AUTOINSTALL_STATE(ACTIVE|INACTIVE)
specifies the status of the program autoinstall function.

ACTIVE
Enable autoinstall for programs, mapsets, and partitionsets.

INACTIVE
Disable autoinstall for programs, mapsets, and partitionsets.

LANGUAGES_AVAILABLE(NO|YES)
specifies whether the autoinstall control program can be called. It can only be called after language establishment.

NO The control program cannot be called.

YES The control program can be called.

RESPONSE and REASON values for SET_AUTOINSTALL

RESPONSE	*REASON*
OK	None
EXCEPTION	None

RESPONSE	*REASON*
DISASTER	None
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

State data access XPI functions

The XPI provides state data access functions that you can use to inquire on, and set, certain system data in the AP domain. These are the DFHAPIQX calls INQ_APPLICATION_DATA, INQUIRE_SYSTEM, and SET_SYSTEM.

The INQ_APPLICATION_DATA call

The INQ_APPLICATION_DATA call enables you to inquire on application system data in the AP domain.

INQ_APPLICATION_DATA

```
DFHAPIQX [CALL,]
   [CLEAR,]
   [IN,
   FUNCTION(INQ_APPLICATION_DATA),]
   [OUT,
   [ACEE(name4 | (Rn) | * ),]     [DSA(name4 | (Rn) | * ),]
   [EIB(name4 | (Rn) | * ),]
   [RSA(name4 | (Rn) | * ),]
   [SYSEIB(name4 | (Rn) | * ),]
   [TCTUA(name4 | (Rn) | * ),]
   [TCTUASIZE(name4 | * ),]
   [TWA(name4 | (Rn) | * ),]
   [TWASIZE(name4 | (Rn) | * ),]
   RESPONSE (name1 | * ),
   REASON (name1 | * )]
```

This command is threadsafe.

ACEE(name4 | (Rn | *)
returns the address of the access control environment element (ACEE).

name4 The name of a fullword area that is to receive the address of the ACEE.

(Rn) A register that is to receive the ACEE address.

* The parameter list itself, in name APIQ_ACEE, is used to hold the address.

DSA(name4 | (Rn | *)
returns the head of the chain of dynamic storage used by application programs to make them reentrant (for example, for assembler programs, the DFHEISTG storage).

name4 The name of a 4-byte area that is to receive the address of the head of the dynamic storage chain.

(Rn) A register that is to receive the DSA address.

* The parameter list itself, in name APIQ_DSA, is used to hold the address.

EIB(name4 | (Rn) | *)
returns the address of the EXEC interface block (EIB) for the current task.

name4 The name of a fullword area that is to receive the address of the EIB.

(Rn) A register that is to receive the address of the EIB.

* The parameter list itself, in name APIQ_EIB, is used to hold the address.

RSA(name4 | (Rn | *)
returns the address of the register save area for the current task.

name4 The name of a fullword area that is to receive the address of the register save area.

(Rn) A register that is to receive the address of the register save area.

* The parameter list itself, name APIQ_RSA, is used to hold the address.

SYSEIB(name4 | (Rn) | *)
returns the address of the system EXEC interface block of the current task.

name4 The name of a fullword area that is to receive the address of the system EXEC interface block.

(Rn) A register that is to receive the address of the system EXEC interface block.

* The parameter list itself, name APIQ_SYSEIB, is used to hold the address.

TCTUA(name4 | (Rn) | *)
returns the address of the terminal control table user area (TCTUA) for the current task.

name4 The name of a fullword area that is to receive the address of the TCTUA.

(Rn) A register that is to receive the address of the TCTUA.

* The parameter list itself, name APIQ_TCTUA, is used to hold the address.

TCTUASIZE(name4 | (Rn) | *)
returns the length in bytes of the TCTUA for the current task.

name4 The name of a 4-byte area that is to receive the length in bytes of the TCTUA.

(Rn) A register that is to receive the length of the TCTUA.

* The parameter list itself, name APIQ_TCTUASIZE, is used to hold the length of the TCTUA.

TWA(name4 | (Rn) | *)
returns the address of the transaction work area.

name4 The name of a fullword area that is to receive the address of the TWA.

(Rn) A register that is to receive the address of the TWA.

* The parameter list itself, name APIQ_TWA, is used to hold the address of the TWA.

TWASIZE(name4 | (Rn) | *)
returns the length, in bytes, of the transaction work area (TWA).

name4 The name of a 4-byte area that is to receive the length, in bytes, of the TWA.

(Rn) A register that is to receive the length of the TWA.

* The parameter list itself, name APIQ_TWASIZE, is used to hold the length of the TWA.

RESPONSE and REASON values for INQ_APPLICATION_DATA

RESPONSE	*REASON*
OK	None
EXCEPTION	DPL_PROGRAM
	NO_TRANSACTION_ENVIRONMENT
	TRANSACTION_DOMAIN_ERROR
DISASTER	ABEND
	LOOP
	INQ_FAILED
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The INQUIRE_SYSTEM call

The INQUIRE_SYSTEM call gives you access to CICS system data in the AP domain.

INQUIRE_SYSTEM

```
DFHSAIQX [CALL,]
   [CLEAR,]
   [IN,
   FUNCTION(INQUIRE_SYSTEM),
   [GMMTEXT(name4),]]
   [OUT,
   [CICSREL(name4 | *),]
   [CICSSTATUS(ACTIVE | FINALQUIESCE |
                  FIRSTQUIESCE| INITIALIZING),]
   [CICSSYS(name1 | *),]
   [CICTSLEVEL(name6 | *),]
   [CWA(name4 | (Rn) | *),]
   [CWALENGTH(name2 | *),]
   [DATE(name4|*),]
   [DTRPRGRM(name8 | *),]
   [GMMLENGTH(name2 | *),]
   [GMMTRANID(name4 | *),]
   [INITSTATUS(FIRSTINIT | INITCOMPLETE | SECONDINIT |
                  THIRDINIT),]
   [JOBNAME(name8 | *),]
   [OPREL(name2 | *),]
   [OPSYS(name1 | *),]
   [OSLEVEL(name4 | *),]
   [PLTPI(name2 | *),]
   [SDTRAN(name4 | *),]
   [SECURITYMGR(EXTSECURITY | NOSECURITY),]
   [SHUTSTATUS(CONTROLSHUT | NOTSHUTDOWN | SHUTDOWN),]
   [STARTUP(COLDSTART | EMERGENCY | WARMSTART),]
   [STARTUPDATE(name4| *),]
   [TERMURM(name8 | *),]
   [TIMEOFDAY(name4| *),]
   [XRFSTATUS(NOXRF | PRIMARY | TAKEOVER),]
   RESPONSE (name1 | * ),
   REASON (name1 | * )]
```

This command is threadsafe.

CICSREL(name4 | *)
returns the level number of the CICS code under which the CICS region is running.

name4 The name of a 4-byte location that is to receive the level number characters as hexadecimal values.

CICSSTATUS(ACTIVE|FINALQUIESE|FIRSTQUIESCE|INITIALIZING)
returns the status of the CICS region.

ACTIVE
The CICS region is active and ready to receive work.

FINALQUIESCE
The CICS region is shutting down, and is in the final stage of quiescing.

FIRSTQUIESCE
The CICS region is shutting down, and is in the first stage of quiescing.

INITIALIZING
The CICS region is initializing.

CICSSYS(name1 | *)
returns the operating system for which the running CICS has been built.

name1 The name of a 1-byte area that is to receive the hexadecimal character of the operating system. A value of "X" represents MVS.

CICSTSLEVEL(name6 | *)
returns the release of CICS Transaction Server under which CICS is running.

name6 The name of a 6-byte area that is to receive the release characters as hexadecimal values.

CWA(name4 | (Rn) | *)
returns the address of the common work area.

name4 The name of a 4-byte field that is to receive the address of the CWA.

(Rn) A register to receive the address of the CWA.

CWALENGTH(name2 | *)
returns the length in bytes of the CWA.

name2 The name of a 2-byte field that is to receive the length of the CWA.

DATE(name4 | *)
returns today's date in packed-decimal form—4-bytes **0Cyyddds**, where:

- **C**is a century indicator. (0=1900, 1=2000, 2=2100, and so on.)
- **yy**=years.
- **ddd**=days.
- **s** is the sign.

name4 The name of a 4-byte location that is to receive the date.

DTRPRGRM(name8 | *)
returns the name of the dynamic routing program.

name8 The name of an 8-byte area that is to receive the name of the dynamic routing program.

GMMLENGTH(name2 | *)
returns the length in bytes of the "good morning" message.

name2 The name of a 2-byte area that is to receive the length of the good morning message.

GMMTEXT(name4)
specifies the address of an area of storage, at least 244 bytes in length and owned by the caller, into which CICS is to return the good morning message.

name4 The address of an area of storage that is to receive the good morning message.

Note: The GMMTEXT parameter must follow the IN statement as an input parameter.

GMMTRANID(name4 | *)
returns the transaction identifier of the CICS good morning transaction.

name4 The name of a 4-byte area that is to receive the CICS good morning transaction id.

INITSTATUS(FIRSTINIT|INITCOMPLETE|SECONDINIT|THIRDINIT)
returns a value indicating the stage reached during CICS initialization.

FIRSTINIT
The first stage of CICS initialization.

INITCOMPLETE
CICS initialization is complete.

SECONDINIT
The second stage of CICS initialization. This stage corresponds to the period when first phase PLTPI programs are run; that is those programs in a PLT that are defined **before** the DFHDELIM statement.

THIRDINIT
The third stage of CICS initialization. This stage corresponds to the period when second phase PLTPI programs are run; that is those programs in a PLT that are defined **after** the DFHDELIM statement.

JOBNAME(name8 | *)
returns the 8-character MVS job name under which the CICS region is running.

name8 The name of a 8-byte area that is to receive the MVS job name.

OPREL(name2 | *)
returns the last 2 digits of the level number of the MVS element of z/OS, under which the CICS region is running.

name2 The name of a 2-byte area that is to receive, as a half-word binary value, the level number of the MVS element of z/OS. For example, z/OS Release 3 MVS is represented by 03.

Note: This field is supported for compatibility purposes only. The information is derived from the last two numbers held in the MVS CVTPRODN field. For example, CVTPRODN holds SP5.2.2 for MVS/ESA SP Version 5 Release 2.2 (in which case OPREL returns 22), and SP6.0.3 for z/OS Release 3. You are recommended to use the OSLEVEL field for the full version and release number of the z/OS product.

OPSYS(name1 | *)
returns the type of operating system on which the CICS regions is running.

name1 The name of a 1-byte area that is to receive the hexadecimal character of the operating system on which CICS is running. A value of "X" represents MVS.

OSLEVEL(name4 | *)
is the version, release, and modification level of the z/OS product on which CICS is running.

name1 The name of a 4-byte area that is to receive the version and release number of z/OS on which CICS is running. A value of "0240" represents z/OS Release 4.

PLTPI(name2 | *)
returns the suffix that identifies the program list table (PLT) containing the list of programs to be run during CICS initialization—the program list table post initialization (PLTPI) list.

name2 The name of a 2-byte area that is to receive the suffix.

SDTRAN(name4 | *)
returns the name of the "shutdown assist" transaction to be run at the beginning of normal or immediate shutdown. The shutdown assist transaction is described in "The shutdown assist utility program, DFHCESD" on page 413.

name4 The name of a 4-byte area to receive the name.

SECURITYMGR(EXTSECURITY|NOSECURITY)
returns a value indicating whether security is active.

EXTSECURITY
CICS is using an external security manager (for example, RACF).

NOSECURITY
Security is not in use in the CICS region—SEC=NO is specified as a system initialization parameter.

SHUTSTATUS(CONTROLSHUT|NOTSHUTDOWN|SHUTDOWN)
returns the shutdown status of the CICS region.

CONTROLSHUT
CICS is performing a controlled shutdown; that is, a normal shutdown with a warm keypoint.

NOTSHUTDOWN
CICS is not in shutdown mode.

SHUTDOWN
CICS is performing an immediate shutdown.

STARTUP(COLDSTART|EMERGENCY|WARMSTART)
returns the type of startup the CICS region performed.

COLDSTART
CICS performed a cold start, either because this was explicitly specified on the system initialization parameter, or because CICS forced a cold start because of the state of the global catalog.

EMERGENCY
CICS performed an emergency restart because the previous run did not shut down normally with a warm keypoint.

WARMSTART
CICS performed a warm restart following the normal shutdown of the previous run.

STARTUPDATE(name4 | *)
returns the start-up-date of this CICS region, in packed decimal form (4-bytes **00yydddc** where **yy**=years, **ddd**=days, **c** is the sign).

name4 The name of a 4-byte location that is to receive the startup date of this CICS system.

TERMURM(name8 | *)
returns the name of the autoinstall user program for terminals.

name8 The name of an 8-byte area that is to receive the name of the autoinstall user program for terminals.

TIMEOFDAY(name4 | *)
returns the current time-of-day in packed decimal form (4-bytes **hhmmsstc** where **hh**=hours, **mm**=minutes, **ss**=seconds, **t**=tenths of a second, and **c** is the sign).

name4 The name of a 4-byte location that is to receive the time.

XRFSTATUS(NOXRF|PRIMARY|TAKEOVER)
returns the XRF status of the CICS region.

NOXRF
CICS was started with the system initialization parameter XRF=NO specified. XRF is not active.

PRIMARY
The CICS region was started as an active CICS in an XRF environment.

TAKEOVER
The CICS region was started as an alternate CICS, with the START=STANDBY system initialization parameter.

RESPONSE and REASON values for INQUIRE_SYSTEM

RESPONSE	*REASON*
OK	None
INVALID	INVALID_FUNCTION
EXCEPTION	LENGTH_ERROR
	UNKNOWN_DATA
DISASTER	INQ_FAILED
PURGED	None

The SET_SYSTEM call

The SET_SYSTEM call allows you to set CICS system data values in the AP domain.

SET_SYSTEM

```
DFHSAIQX [CALL,]
   [CLEAR,]
   [IN,
   FUNCTION(SET_SYSTEM),
   [DTRPRGRM(name8 | string | 'string'),]
   [GMMLENGTH(name2 | (Rn) | expression),]
   [GMMTEXT(name8 | (Rn)),]]
   [OUT,
   RESPONSE (name1 | * ),
   REASON (name1 | * )]
```

This command is threadsafe.

DTRPRGRM(name8 | string | 'string')
specifies the name of the dynamic routing program.

name8 The name of an 8-byte area that contains the name of the dynamic routing program.

string A string of character, without intervening blanks, that defines the name of the dynamic routing program being set.

'string'
A string of character without intervening blanks. If you want to document a name (label) in your program, use this form.

GMMLENGTH(name2 | (Rn))
specifies the length of the new "good morning" message supplied by the GMMTEXT parameter.

name2 The name of a 2-byte area that contains, as a half-word binary value, the length of the new good morning message.

(Rn) A register that contains the length of the new good morning message.

GMMTEXT(name4 | (Rn))
specifies the new good morning message.

name4 The name of a 4-byte location that contains the address of a storage area (up to a maximum of 246 bytes long) that contains the good morning message.

(Rn) A register that contains the address of a storage area (up to a maximum of 246 bytes long) that contains the good morning message.

RESPONSE and REASON values for SET_SYSTEM

RESPONSE	*REASON*
OK	None
INVALID	INVALID_FUNCTION
EXCEPTION	AKP_SIZE_ERROR
	NO_KEYPOINT
DISASTER	SET_FAILED
PURGED	None

Storage control XPI functions

The XPI provides seven storage control functions. These are the DFHSMMCX macro calls GETMAIN, FREEMAIN, INQUIRE_ELEMENT_LENGTH, and INQUIRE_TASK_STORAGE, and the DFHSMSRX calls INQUIRE_ACCESS, INQUIRE_SHORT_ON_STORAGE, and SWITCH_SUBSPACE.

DFHSMMCX calls cannot be used in any exit program invoked from any global user exit point in the following domains or program:

- Dispatcher domain
- Dump domain
- Monitor domain
- Statistics domain
- Transient data program.

The GETMAIN call

GETMAIN acquires an element of storage for use by your exit program. You can ask for a specific CLASS of storage, and you can request that it is initialized to a single-byte value.

Storage that is acquired by using a GETMAIN call and that is in the following classes is released by CICS when the TCA being used at the time of the acquisition terminates:

- CICS
- CICS24
- USER
- USER24.

In contrast, storage in the following classes is not released automatically at task-end. You must use the FREEMAIN call to release it.

- SHARED_CICS
- SHARED_CICS24
- SHARED_USER
- SHARED_USER24
- TERMINAL.

Also, some user exits can be invoked from system tasks. In these circumstances, storage is not released until the next CICS shutdown. Therefore, use the FREEMAIN call to release all storage areas acquired by a GETMAIN call as soon as you finish using them.

GETMAIN

```
DFHSMMCX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(GETMAIN),
      GET_LENGTH(name4 | (Rn) | expression),
      STORAGE_CLASS(CICS|CICS24|SHARED_CICS|SHARED_CICS24|
                  SHARED_USER|SHARED_USER24|USER|USER24|TERMINAL),
      SUSPEND(NO|YES),
      [INITIAL_IMAGE(name1 | literalconst),]
      [TCTTE_ADDRESS(name4 | (Ra)),]]
      [OUT,
      ADDRESS(name4 | (Rn) | *),
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

ADDRESS(name4 | (Rn) | *)
Returns the address of the storage obtained by the call.

name4 The name of a fullword where the obtained storage address is saved.

(Rn) A register that is set to point to the obtained storage.

* The parameter list itself, name SMMC_ADDRESS, is used to keep the address.

GET_LENGTH(name4 | (Rn) | expression)
Specifies the number of bytes of storage you want, expressed in any of the following ways:

name4 The name of a fullword specifying, in binary, the number of bytes.

(Rn) A register containing, in binary, the number of bytes.

expression
A valid assembler-language expression; for example, a number, a symbolic expression, or a combination of the two.

If you request TERMINAL storage, the length you specify does not include the length of the storage accounting area (SAA). The maximum length you can specify is 65,515 bytes. CICS storage management adds an 8-byte SAA, and the address returned by the XPI call is that of the start of the SAA.

If you request CICS24, CICS, USER24, USER, SHARED_CICS24, SHARED_CICS, SHARED_USER24, or SHARED_USER storage, you need only specify the length needed by your program. The address returned is that of the start of your data storage. The maximum size of storage for these storage classes is the same as the size of the DSA from which they are allocated.

INITIAL_IMAGE(name1 | literalconst)
Specifies the initializing pattern. For example, you might want to set the acquired storage to binary zeros.

name1 The name of a location where the one-byte initializing pattern is stored.

literalconst
A number in the form of a literal, for example B'00000000', X'FF', X'FC', "0", or an equate symbol with a similar value.

STORAGE_CLASS(CICS|CICS24|SHARED_CICS|SHARED_CICS24|SHARED_USER|SHARED_USER24|USER|USER24|TERMINAL)
Specifies the class of the storage that is the subject of the call. The values you can assign to this option, and the type of storage each represents, are listed in Table 47.

Table 47. CICS storage classes

STORAGE_CLASS	Type of storage
CICS	Task-lifetime CICS-key storage above 16 MB but below 2 GB
CICS24	Task-lifetime CICS-key storage below 16 MB
SHARED_CICS	Shared CICS-key storage above 16 MB but below 2 GB
SHARED_CICS24	Shared CICS-key storage below 16 MB
SHARED_USER	Shared user-key storage above 16 MB but below 2 GB
SHARED_USER24	Shared user-key storage below 16 MB
TERMINAL	This class of storage has an 8-byte SAA.
USER	Task-lifetime user-key storage above 16 MB but below 2 GB
USER24	Task-lifetime user-key storage below 16 MB

You must specify a storage class on a GETMAIN request. On a FREEMAIN request it is an optional parameter, and any value that you specify is not checked by CICS.

SUSPEND(YES|NO)
Specifies whether to suspend your request if there is less storage available than you requested in the GET_LENGTH option.

TCTTE_ADDRESS(name4 | (Ra))
Specifies the address of the terminal control table terminal entry (TCTTE). On

GETMAIN requests, you must code this option if you specify a class of TERMINAL on the STORAGE_CLASS option. On FREEMAIN requests, you must code this option if you release TERMINAL-class storage.

Note: Before you obtain TERMINAL class storage, check TCAFCI bit 7 to ensure that the TCA is running under a terminal.

name4 The name of a fullword containing the address.

(Ra) A register that points to the TCTTE.

RESPONSE and REASON values for GETMAIN

RESPONSE	*REASON*
OK	None
EXCEPTION	INSUFFICIENT_STORAGE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.
2. INSUFFICIENT_STORAGE is returned if the GETMAIN request was specified with SUSPEND(NO), and there was not enough storage available to satisfy the request.
3. PURGED is returned if the GETMAIN request was specified with SUSPEND(YES), there was not enough storage to satisfy the request, and the task was purged.

The FREEMAIN call

FREEMAIN releases an area of storage that is currently allocated to your exit program.

FREEMAIN

```
DFHSMMCX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(FREEMAIN),
      ADDRESS(name4 | (Rn) | *),
      [STORAGE_CLASS(CICS|CICS24|SHARED_CICS|SHARED_CICS24|
                  SHARED_USER|SHARED_USER24|USER|USER24|TERMINAL),]
      [TCTTE_ADDRESS(pointer),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

For an explanation of the options, see "The GETMAIN call" on page 845.

RESPONSE and REASON values for FREEMAIN

RESPONSE	*REASON*
OK	None
EXCEPTION	None

RESPONSE	*REASON*
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

The INQUIRE_ACCESS call

INQUIRE_ACCESS returns the access-key of an element of storage specified by start address and length. If the element is not wholly contained within one of the CICS dynamic storage areas (DSAs), CICS returns an exception response.

INQUIRE_ACCESS

```
DFHSMSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_ACCESS),
      ELEMENT_ADDRESS(name4 | (Rn) | *),
      ELEMENT_LENGTH(name4 | (Rn) | *),]
      [OUT,
      ACCESS(CICS | READ_ONLY | USER),
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

ACCESS(CICS|READ_ONLY|USER)
returns the access-key of the storage element.

CICS CICS-key

READ_ONLY
Readonly storage

USER User-key.

ELEMENT_ADDRESS(name4 | (Rn) | *)
specifies the address of the storage element.

ELEMENT_LENGTH(name4 | (Rn) | *)
specifies the length of the storage element, in bytes. A length of zero is treated as a length of one.

RESPONSE and REASON values for INQUIRE_ACCESS

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_ELEMENT
DISASTER	None
INVALID	None
KERNERROR	None

The INQUIRE_ELEMENT_LENGTH call

INQUIRE_ELEMENT_LENGTH enables you to pass the address of any part of an element of task-lifetime storage, and to obtain from CICS the start address and the length of the storage element that contains the passed address.

INQUIRE_ELEMENT_LENGTH

```
DFHSMMCX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION (INQUIRE_ELEMENT_LENGTH),
       ADDRESS (name4 | (Rn) | *),]
       [OUT,
       ELEMENT_ADDRESS(name4 | (Rn) | *),
       ELEMENT_LENGTH(name4 | (Rn) | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

ADDRESS(name4 | (Rn) | *)
specifies an address that lies within an element of task-lifetime storage of the current task.

CICS accepts addresses that reference the leading or trailing check zones as being valid addresses for the element of storage you are inquiring upon.

ELEMENT_ADDRESS(name4 | (Rn) | *)
returns the start address of the element of task-lifetime storage referenced by the ADDRESS parameter. The start address returned does **not** include the leading check zone.

ELEMENT_LENGTH(name4 | (Rn) | *)
returns the length of the element of task-lifetime storage referenced by the ADDRESS parameter. The length returned does **not** include the leading or trailing check zones.

RESPONSE and REASON values for INQUIRE_ELEMENT_LENGTH

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_ADDRESS
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The INQUIRE_SHORT_ON_STORAGE call

INQUIRE_SHORT_ON_STORAGE enables you to determine whether CICS is short on 64-bit (above-the-bar) storage, short on storage above 16 MB but below 2 GB (above the line), or short on storage below 16 MB (below the line).

INQUIRE_SHORT_ON_STORAGE

```
DFHSMSRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_SHORT_ON_STORAGE),]
       [OUT,
       SOS_ABOVE_THE_BAR(NO|YES),
       SOS_ABOVE_THE_LINE(NO|YES),
       SOS_BELOW_THE_LINE(NO|YES),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

SOS_ABOVE_THE_BAR(NO|YES),
Returns YES if CICS is currently short on 64-bit (above-the-bar) storage, and NO if not.

SOS_ABOVE_THE_LINE(NO|YES),
Returns YES if CICS is currently short on storage above 16 MB but below 2 GB, and NO if not.

SOS_BELOW_THE_LINE(NO|YES),
returns YES if CICS is currently short on storage below 16 MB, and NO if not.

RESPONSE and REASON values for INQUIRE_SHORT_ON_STORAGE

RESPONSE	*REASON*
OK	None
DISASTER	None
KERNERROR	None

The INQUIRE_TASK_STORAGE call

INQUIRE_TASK_STORAGE enables you to request details of all elements of task-lifetime storage belonging to a task. You can specify the transaction number of the task explicitly on the call, or let it default to the current task.

INQUIRE_TASK_STORAGE

```
DFHSMMCX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION (INQUIRE_TASK_STORAGE),
       [TRANSACTION_NUMBER(name4 | (Rn) | *),]
       ELEMENT_BUFFER(buffer-descriptor),
       LENGTH_BUFFER(buffer-descriptor),]
       [OUT,
       NUMBER_OF_ELEMENTS(name4 | (Rn) | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

ELEMENT_BUFFER(buffer-descriptor)
defines the address and length of a buffer into which CICS returns a list of start addresses of all the elements of task-lifetime storage belonging to either the specified task or, by default, the current task.

The start addresses returned do **not** include the leading check zone. For a description of a buffer descriptor, see buffer-descriptor.

LENGTH_BUFFER(buffer-descriptor)
defines the address and length of a buffer into which CICS returns a list of the lengths of the elements of task-lifetime storage belonging to either the specified task or, by default, the current task. The lengths returned do **not** include the leading or trailing check zones.

For a description of a buffer descriptor, see buffer-descriptor.

NUMBER_OF_ELEMENTS(name4 | (Rn) | *)
returns the number of entries in each of the two buffers, ELEMENT_BUFFER and LENGTH_BUFFER, as a full-word binary value.

TRANSACTION_NUMBER(name4 | (Rn) | *)
specifies, as a 4 byte packed decimal value, the transaction number of the task to whom the storage belongs.

If you omit the transaction (task) number, CICS assumes the current task.

RESPONSE and REASON values for INQUIRE_TASK_STORAGE

RESPONSE	*REASON*
OK	None
EXCEPTION	INSUFFICIENT_STORAGE NO_TRANSACTION_ENVIRONMENT
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The SWITCH_SUBSPACE call

SWITCH_SUBSPACE causes CICS to switch from a subspace to base space, if the task is not already executing in the base space. If the task is already in the base space, storage manager ignores the call.

This function can be used by global user exit programs that receive control in subspace and for some reason need to switch into basespace.

SWITCH_SUBSPACE

```
DFHSMSRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(SWITCH_SUBSPACE),
       SPACE(BASESPACE),]
       [OUT,
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

SPACE(BASESPACE)
specifies that CICS is to switch the task issuing the call to the basespace, if it is currently executing within a subspace. This enables the task to read and write to another task's user-key task-lifetime storage.

RESPONSE and REASON values for SWITCH_SUBSPACE

RESPONSE	*REASON*
OK	None
DISASTER	None
KERNERROR	None

Trace control XPI function

The XPI provides one trace control function. This is the DFHTRPTX call TRACE_PUT.

Restriction: DFHTRPTX calls cannot be used in any exit program invoked from any global user exit point in the:

- Dispatcher domain
- Dump domain
- Monitor domain
- Statistics domain
- Transient data program.

The TRACE_PUT call

TRACE_PUT writes a trace entry to the active trace destinations.

Only make a TRACE_PUT call when UEPTRON indicates that tracing is active for the function containing the exit program (see UEPTRON in DFHUEPAR). You might prefer to make exception trace entries, in case of serious errors, without testing UEPTRON.

If you use TRACE_PUT to write exception trace entries, identify these so they are highlighted as exception trace entries by the trace formatting utility program. To identify an exception trace entry, enter the literal string 'USEREXC' in the DATA1 block descriptor field on the DFHTRPTX call.

TRACE_PUT

```
DFHTRPTX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(TRACE_PUT),
      POINT_ID(literalconst | name2 | (Rn)),
      [DATA1(block-descriptor),]
      [DATA2(block-descriptor),]
      [DATA3(block-descriptor),]
      [DATA4(block-descriptor),]
      [DATA5(block-descriptor),]
      [DATA6(block-descriptor),]
      [DATA7(block-descriptor),]
      [RETURN_ADDR(expression | name4 | (Ra)),]]
      [OUT,
      RESPONSE(name1 | *)]
```

This command is threadsafe.

DATA*n*(block-descriptor)
Specifies up to seven areas to be included in the data section of the trace entry. For a description of valid block-descriptors, see "XPI syntax" on page 393. If you specify any given DATA*n*, then DATA1 through DATA(*n*-1) must be coded before DATA*n*. The specified DATA items are printed in the trace output in the order specified, that is, in order of DATA1 through DATA*n*. A 2-byte length field is printed before the data field itself. The maximum total length of the data that can be traced in one call is 4000 bytes. The total length of all the data fields and all their 2-byte length fields must be within this limit.

POINT_ID(literalconst|name2|(Rn))
Specifies the trace entries made as a result of this request. Every TRACE_PUT call within a calling domain should specify a unique POINT_ID. This enables you to locate the origin of a trace call when examining a formatted trace. The POINT_IDs must be in the range decimal 256 through 511 (X'100' through X'1FF'). This range is not used in CICS modules, but is reserved for user exits.

literalconst
A number in the form of a literal, containing the ID

name2 The name of a 2-byte field containing the ID

(Rn) A register, the two low-order bytes of which contain the ID.

RETURN_ADDR(expression|name4|(Ra))
Specifies the value that appears in the return address field of the trace entry.

expression
A valid assembler-language expression that results in the address

name4 The name of a fullword containing the address

(Ra) A register containing the address.

Transaction management XPI functions

The XPI provides transaction management functions that you can use to inquire about transactions and set certain transaction parameters.

The INQUIRE_CONTEXT call

INQUIRE_CONTEXT returns information about the environment in which a transaction is running. Specifically, it provides information for transactions running in a bridge environment.

INQUIRE_CONTEXT

```
DFHBRIQX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_CONTEXT),]
       [OUT,
       [CONTEXT(byte1),]
       [BRIDGE_TRANSACTION_ID(name4),]
       [BRIDGE_EXIT_PROGRAM(name8),]
       [BFB_TOKEN(name4),]
       [BRXA_TOKEN(name4),]
       [FACILITYTOKEN(name8),]
       [START_TYPE(byte1),]
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

BFB_TOKEN(name4)
returns a pointer that contains the address of the bridge facility used by this task. Although the bridge facility is not a real terminal, it is represented by a data structure that has the same format as a TCTTE and can be mapped using the DSECT DFHTCTTE. If CONTEXT returns NORMAL, the contents of this field are meaningless.

Note: In earlier releases of CICS, this field was called BRIDGE_FACILITY_TOKEN.

name4 The name of a 4-byte location to receive the token.

BRIDGE_EXIT_PROGRAM(name8)
returns the name of the bridge exit program used by this task. If CONTEXT returns NORMAL, the contents of this field are meaningless.

name8 The name of an 8-byte location to receive the name of the bridge exit program.

BRIDGE_TRANSACTION_ID(name4)
returns the name of the bridge monitor transaction that issued a START

BREXIT TRANSID command to start this transaction. If CONTEXT returns NORMAL, the contents of this field are meaningless.

name4 The name of a 4-byte location to receive the name of the bridge monitor transaction.

BRXA_TOKEN(name4)
returns a token that contains the address of the bridge exit area (BRXA) used by this task. The BRXA is not applicable to the Link3270 bridge (START_TYPE=BRIQ_LINK). The format of BRXA is defined by the DFHBRARx copy books. If CONTEXT returns NORMAL, the contents of this field are meaningless.

name4 The name of a 4-byte location to receive the token.

CONTEXT(byte1)
returns, in a 1-byte location (*byte1*), the type of environment in which the transaction is running.

BRIDGE
A user transaction that was started using a bridge

BREXIT
A bridge exit program

NORMAL
A transaction that is not running in a bridge environment.

FACILITYTOKEN(name8)
returns the facilitytoken (an identifier associated with the bridge facility). If CONTEXT returns NORMAL, the contents of this field are meaningless.

name8 The name of an 8-byte location to receive the facilitytoken.

START_TYPE(byte1)
returns, in a 1–byte location (*byte1*), an indication of how the 3270 bridge was started. If CONTEXT returns NORMAL, the contents of this field are meaningless.

BRIQ_START
The bridge was started using START BREXIT.

BRIQ_LINK
The bridge was started using the Link3270 mechanism.

RESPONSE and REASON values for INQUIRE_CONTEXT

RESPONSE	*REASON*
OK	None
DISASTER	ABEND
	LOOP
INVALID	None
EXCEPTION	NO_TRANSACTION_ENVIRONMENT
KERNERROR	None

The INQUIRE_DTRTRAN call

INQUIRE_DTRTRAN returns the name of the dynamic transaction routing (DTR) transaction definition.

The DTR transaction definition provides common attributes for transactions that are to be dynamically routed and which do not have a specific transaction

definition. It is specified on the DTRTRAN system initialization parameter; the CICS-supplied default definition is CRTX.

INQUIRE_DTRTRAN

```
DFHXMSRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_DTRTRAN),]
       [OUT,
       DTRTRAN(name4),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

DTRTRAN(name4)
returns the name of the DTR transaction definition to used for routing transactions that are not defined by an explicit transaction resource definition.

name4 The name of a 4-byte location that is to receive the name of the DTR transaction definition. If 'NO' was specified on the DTRTRAN system initialization parameter, 'NO' will be placed in this field.

RESPONSE and REASON values for INQUIRE_DTRTRAN

RESPONSE	*REASON*
OK	None
DISASTER	ABEND
	LOGIC_ERROR
	LOOP
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The INQUIRE_MXT call

The INQUIRE_MXT function is provided on the DFHXMSRX macro call. Its purpose is to provide current value of the MXT parameter.

INQUIRE_MXT

```
DFHXMSRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_MXT),]
       [OUT,
       CURRENT_ACTIVE(name4 | (Rn) ),
       MXT_LIMIT(name4 | (Rn)),
       MXT_QUEUED(name4 | (Rn) ),
       TCLASS_QUEUED(name4 | (Rn) ),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

CURRENT_ACTIVE(name4 | (Rn))
returns the current number of all active user tasks.

name4 The name of a 4-byte location that is to receive the current number of active user tasks, expressed as a binary value.

(Rn) A register to receive the current number of active user tasks, expressed as a binary value.

MXT_LIMIT(name4 | (Rn))
returns the current number of the MXT parameter.

name4 The name of a 4-byte location that is to receive the maximum number of all user tasks currently allowed, expressed as a binary value.

(Rn) A register to receive the maximum number of all tasks currently allowed, expressed as a binary value.

MXT_QUEUED(name4 | (Rn))
returns the current number of user transactions that are queued as a result of the maximum tasks (MXT) being reached.

name4 The name of a 4-byte location that is to receive the current number of queued user tasks, expressed as a binary value.

(Rn) A register to receive the current number of queued user tasks, expressed as a binary value.

TCLASS_QUEUED(name4 | (Rn))
returns the current number of all transactions that are queued for transaction class membership.

name4 The name of a 4-byte location that is to receive the current number of queued transaction class members, expressed as a binary value.

(Rn) A register to receive the current number of queued transaction class members, expressed as a binary value.

RESPONSE and REASON values for INQUIRE_MXT

RESPONSE	*REASON*
OK	None
DISASTER	LOGIC_ERROR
	ABEND
	LOOP
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The INQUIRE_TCLASS call

The INQUIRE_TCLASS function is provided on the DFHXMCLX macro call. Its purpose is to provide current information about the specified transaction class (TCLASS).

INQUIRE_TCLASS

```
DFHXMCLX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_TCLASS),
       INQ_TCLASS_NAME(name8 | string | 'string'),]
       [OUT,
       [CURRENT_ACTIVE(name4 | (Rn)),]
       [CURRENT_QUEUED(name4 | (Rn)),]
       [MAX_ACTIVE(name4 | (Rn)),]
       [PURGE_THRESHOLD(name4 | (Rn)),]
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

CURRENT_ACTIVE(name4 | (Rn))
returns the current number of active user tasks in this transaction class.

name4 The name of a 4-byte location that is to receive the current number of active user tasks for this transaction class, expressed as a binary value.

(Rn) A register to receive the current number of active user tasks for this transaction class, expressed as a binary value.

CURRENT_QUEUED(name4 | (Rn))
returns the current number of queued user tasks.

name4 The name of a 4-byte location that is to receive the current number of queued user tasks in this transaction class, expressed as a binary value.

(Rn) A register to receive the current number of queued user tasks, expressed as a binary value.

INQ_TCLASS_NAME(name8 | string | 'string')
specifies the name of the transaction class for this inquiry.

name8 The name of an 8-byte location that contains the name of the transaction class being inquired on.

string A string of characters, without intervening blanks, naming the transaction class.

'string'
A string of characters, within quotation marks, naming the transaction class. The string length is set to 8 by padding with blanks within the quotation marks.

MAX_ACTIVE(name4 | (Rn))
returns the current maximum number of active tasks allowed for the transaction class.

name4 The name of a 4-byte location that is to receive the current maximum number of active tasks currently allowed for this transaction class, expressed as a binary value.

(Rn) A register to receive the current maximum number of active tasks currently allowed for this transaction class, expressed as a binary value.

PURGE_THRESHOLD(name4 | (Rn))
returns the purge threshold limit for this transaction class.

name4 The name of a 4-byte location that is to receive the current purge threshold limit for this transaction class, expressed as a binary value.

(Rn) A register to receive the current purge threshold limit for this transaction class, expressed as a binary value.

RESPONSE and REASON values for INQUIRE_TCLASS

RESPONSE	*REASON*
OK	None
DISASTER	LOGIC_ERROR
INVALID	None
EXCEPTION	UNKNOWN_CLASS

The INQUIRE_TRANDEF call

The INQUIRE_TRANDEF function is provided on the DFHXMXDX macro call. Its purpose is to allow you to obtain information about the specified transaction definition. In general, this function call is equivalent to the EXEC CICS INQUIRE TRANSACTION command, with some differences.

INQUIRE_TRANDEF

```
DFHXMXDX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_TRANDEF),
       INQ_TRANSACTION_ID(name4 | string | 'string'),]
       [OUT,
       [BREXIT(name8),]
       [CMDSEC(name1),]
       [DTIMEOUT(name4 | (Rn)),]
       [DUMP(name1),]
       [DYNAMIC(name1),]
       [INDOUBT(name1),]
       [INDOUBT_WAIT(name1),]
       [INDOUBT_WAIT_TIME(name4),]
       [INITIAL_PROGRAM(name8),]
       [ISOLATE(name1),]
       [LOCAL_QUEUING(name1),]
       [OTSTIMEOUT(name4 | (Rn)),]
       [PARTITIONSET(name1),]
       [PARTITIONSET_NAME(name8),]
       [PROFILE_NAME(name8),]
       [REMOTE(name1),]
       [REMOTE_NAME(name8),]
       [REMOTE_SYSTEM(name4),]
       [RESSEC(name1),]
       [RESTART(name1),]
       [ROUTABLE_STATUS(ROUTABLE|NOT_ROUTABLE),]
       [RUNAWAY_LIMIT(name4 | (Rn)),]
       [SHUTDOWN(name1),]
       [SPURGE(name1),]
       [STATUS(name1),]
       [STORAGE_CLEAR(name1),]
       [STORAGE_FREEZE(name1),]
       [SYSTEM_ATTACH(name1),]
       [SYSTEM_RUNAWAY(name1),]
       [TASKDATAKEY(name1),]
       [TASKDATALOC(name1),]
       [TCLASS(name1),[TCLASS_NAME(name8),]]
       [TPURGE(name1),]
       [TRACE(name1),]
       [TRAN_PRIORITY(name4 | (Rn)),]
       [TRAN_ROUTING_PROFILE(name8),]
       [TRANSACTION_ID(name4),]
       [TWASIZE(name4 | (Rn)),]
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

The following parameter descriptions explain briefly the possible values that can be returned on an INQUIRE_TRANDEF call. For a more detailed explanation of some of these parameters, see the corresponding parameter descriptions for the TRANSACTION resource definition in .

BREXIT(name8)
returns the name of the default bridge exit program specified for the named transaction. If no bridge exit is specified, blanks are returned.

name8 The name of an 8-byte location to receive the name of the bridge exit program.

CMDSEC(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether command security checking is required for the transaction.

XMXD_YES
Command security checking is required.

XMXD_NO
Command security checking is not required.

DTIMEOUT(name4)
returns the deadlock time-out value for the transaction.

name4 The name of a 4-byte location that is to receive the deadlock time-out value, expressed as a binary value.

(Rn) A register to receive the deadlock time-out value, expressed as a binary value.

Note that a value of zero means that the transaction resource definition specifies DTIMOUT(NO).

DUMP(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether CICS is to take a transaction dump if the transaction abends.

XMXD_YES
A transaction dump is required.

XMXD_NO
A transaction dump is not required.

DYNAMIC(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction is defined for dynamic transaction routing.

XMXD_YES
The transaction is to be dynamically routed to a remote CICS.

XMXD_NO
The transaction is not to be dynamically routed.

INDOUBT(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the action to be taken if the CICS region fails or loses connectivity with its coordinator while a unit of work is in the indoubt period. (The action is based on the ACTION attribute of the TRANSACTION resource definition.)

The action is dependent on the values returned in INDOUBT_WAIT and INDOUBT_WAIT_TIME; if INDOUBT_WAIT returns XMXD_YES, the action is not taken until the time returned in INDOUBT_WAIT_TIME expires.

XMXD_BACKOUT
Any changes made by the transaction to recoverable resources are to be backed out.

XMXD_COMMIT
Any changes made by the transaction to recoverable resources are to be committed.

INDOUBT_WAIT(name1)
returns, in a 1-byte location (*name1*), an equated value indicating how a unit of work (UOW) is to respond if a failure occurs while it is in an indoubt state.

XMXD_NO
The UOW is not to wait, pending recovery from the failure. CICS is to take immediately whatever action is specified on the ACTION attribute of the TRANSACTION definition.

XMXD_YES
The UOW is to wait, pending recovery from the failure, to determine whether recoverable resources are to be backed out or committed.

INDOUBT_WAIT_TIME(name4)
returns the length of time, in minutes, after a failure during the indoubt period, before the transaction is to take the action returned in the INDOUBT field. The returned value is valid only if the unit of work is indoubt and INDOUBT_WAIT returns XMXD_YES.

name4 The name of a 4-byte location that is to receive the delay time, expressed as a binary value.

See also INDOUBT and INDOUBT_WAIT.

INITIAL_PROGRAM(name8)
returns the name of the initial program to be given control for the transaction.

name8 The name of an 8-byte location to receive the initial program name.

INQ_TRANSACTION_ID(name4 | string | 'string')
specifies the transaction identifier for this transaction definition inquiry.

name4 The name of a 4-byte location that contains the name of the transaction identifier.

string A string of characters, without intervening blanks, naming the transaction identifier.

'string'
A string of characters, within quotation marks, naming the transaction identifier. The string length is set to 4 by padding with blanks within the quotation marks.

ISOLATE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether transaction isolation is required for the transaction's task-lifetime user-key storage.

XMXD_NO
Transaction isolation is not required for task-lifetime user-key storage.

XMXD_YES
Transaction isolation is required for task-lifetime user-key storage.

LOCAL_QUEUING(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether a start request for this transaction is eligible to queue locally if the transaction is to be started on another system, and the remote system is not available.

XMXD_NO
The request is not to be queued locally.

XMXD_YES
The request can be queued locally.

OTSTIMEOUT(name4)
returns the default period in seconds that an Object Transaction Service (OTS) transaction created in an Enterprise JavaBeans (EJB) environment and executing under this CICS transaction is allowed to execute without the initiator of the OTS transaction taking a syncpoint (or rolling back the OTS transaction).

name4 The name of a 4-byte location to receive the timeout setting, expressed as a binary value.

(Rn) A register to receive the timeout setting, expressed as a binary value.

A value of zero means that the transaction resource definition specifies OTSTIMEOUT(NO).

PARTITIONSET(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the partitionset specified on the transaction definition.

XMXD_KEEP
The reserved name KEEP is specified for the partitionset, which means tasks running under this transaction definition use the application partitionset for the terminal associated with the transaction.

XMXD_NAMED
The partitionset is named specifically on the transaction definition. The name is returned on the PARTITIONSET_NAME parameter.

XMXD_NONE
There is no partitionset specified for the transaction definition.

XMXD_OWN
The reserved name OWN is specified for the partitionset, which means tasks running under this transaction definition perform their own partitionset management.

PARTITIONSET_NAME(name8)
returns the name of the partitionset defined on the transaction definition.

name8 The name of an 8-byte location that is to receive the name of the partitionset.

PROFILE_NAME(name8)
returns the name of the profile definition that is associated with the transaction definition.

name8 The name of an 8-byte location to receive the name of the profile definition associated with the transaction definition.

REMOTE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction is defined as remote.

XMXD_NO
The transaction is not a remote transaction.

XMXD_YES
The transaction is a remote transaction.

REMOTE_NAME(name8)
returns the name by which the transaction is known in a remote system.

name8 The name of an 8-byte location to receive the transaction's remote name.

REMOTE_SYSTEM(name4)
returns the name of the remote system as specified on the transaction definition.

If the DYNAMIC parameter returns XMXD_YES, REMOTE_SYSTEM returns the default name, which can be changed by the dynamic routing program.

If the DYNAMIC parameter returns XMXD_NO, this is the actual remote system to which the transaction is to be routed.

name4 The name of a 4-byte location to receive the defined name of the remote system.

RESSEC(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether resource security checking is required for the transaction.

XMXD_NO
Resource security checking is not required.

XMXD_YES
Resource security checking is required.

RESTART(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction is to be considered for transaction restart.

XMXD_NO
The transaction cannot be restarted.

XMXD_YES
The transaction can be restarted.

ROUTABLE_STATUS(ROUTABLE|NOT_ROUTABLE)
returns a value indicating whether, if the transaction is the subject of an eligible EXEC CICS START command, it will be routed using the enhanced routing method.

NOT_ROUTABLE
If the transaction is the subject of a START command, it will be routed using the "traditional" method.

ROUTABLE
If the transaction is the subject of an eligible START command, it will be routed using the enhanced method.

For details of the enhanced and "traditional" methods of routing transactions invoked by EXEC CICS START commands, see the *CICS Intercommunication Guide*.

RUNAWAY_LIMIT(name4 | (Rn))
returns the runaway-task time limit specified on the transaction definition. If SYSTEM_RUNAWAY is XMXD_YES, the value returned is the value defined by the **ICVR** system initialization parameter.

name4 The name of a 4-byte location that is to receive the task runaway limit, expressed as a binary value.

(Rn) A register to receive the task runaway limit, expressed as a binary value.

SHUTDOWN(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction can be run during CICS shutdown.

XMXD_DISABLED
The transaction is disabled from running during CICS shutdown.

XMXD_ENABLED
The transaction is enabled to run during CICS shutdown.

SPURGE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction is defined as system-purgeable.

XMXD_NO
The transaction is not system-purgeable.

XMXD_YES
The transaction is system-purgeable.

STATUS(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the status of the transaction definition.

XMXD_DISABLED
The transaction definition is disabled.

XMXD_ENABLED
The transaction definition is enabled.

STORAGE_CLEAR(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether task-lifetime storage, of tasks associated with this transaction definition, is to be cleared before it is freed by a FREEMAIN command.

XMXD_NO
Task-lifetime storage need not be cleared before it's freed.

XMXD_YES
Task-lifetime storage must be cleared before it's freed.

STORAGE_FREEZE(name1 | (Rn))
returns, in a 1-byte location (*name1*), an equated value indicating whether storage freeze is defined for the transaction by means of the STGFRZ option on the CICS-supplied field engineering transaction, CSFE.

XMXD_NO
Storage is freed normally during the running of the transaction.

XMXD_YES
Storage that is normally freed during the running of a transaction is frozen.

SYSTEM_ATTACH(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the tasks attached with this tranid are always to be attached as system tasks.

XMXD_NO
A user task is being attached for this transaction.

XMXD_YES
A system task is being attached for this transaction.

SYSTEM_RUNAWAY(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction definition specifies the system default runaway-task time limit, which is specified on the **ICVR** system initialization parameter.

XMXD_NO
The transaction is not governed by the system runaway limit.

XMXD_YES
The transaction definition specifies the system default runaway limit.

TASKDATAKEY(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the storage key of task-lifetime storage for tasks associated with this transaction definition.

XMXD_CICS
CICS key is specified for task-lifetime storage.

XMXD_USER
USER key is specified for task-lifetime storage.

TASKDATALOC(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the data location of task-lifetime storage for tasks associated with this transaction definition.

XMXD_ANY
Task-lifetime storage can be located above 16 MB in virtual storage.

XMXD_BELOW
Task-lifetime storage must be located below 16 MB in virtual storage.

TCLASS(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction belongs to a transaction class.

XMXD_NO
The transaction is not a member of a transaction class.

XMXD_YES
The transaction is a member of the transaction class named in the TCLASS_NAME parameter.

TCLASS_NAME(name8)
returns the name of the transaction class to which the transaction belongs.

name8 The name of an 8-byte location to receive transaction class name to which the transaction belongs.

TPURGE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction is defined as purgeable in the event of a z/OS Communications Server terminal error.

XMXD_NO
The transaction can not be purged if a terminal error occurs.

XMXD_YES
The transaction can be purged if a terminal error occurs.

TRACE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the level of tracing defined for the transaction:

XMXD_SPECIAL
CICS special-level trace This is the result of special trace being set by means of an EXEC CICS SET TRANSACTION command.

XMXD_STANDARD
CICS standard-level trace This equates to TRACE(YES) in the TRANSACTION resource definition.

XMXD_SUPPRESSED
Tracing is suppressed for the transaction This equates to TRACE(NO) in the TRANSACTION resource definition.

TRAN_PRIORITY(name4 | (Rn))
returns the transaction priority specified on the transaction definition.

name4 The name of a 4-byte location to receive the transaction priority, expressed as a binary value.

(Rn) A register to receive the transaction priority, expressed as a binary value.

TRAN_ROUTING_PROFILE(name8)
returns the name of the profile that CICS is to use to route the transaction to a remote system.

name8 The name of an 8-byte location to receive the transaction-routing profile.

TRANSACTION_ID(name4)
returns the primary transaction identifier for this transaction definition inquiry.

name4 The name of a 4-byte location that contains the name of the transaction identifier.

TWASIZE(name4 | (Rn))
returns the size of the transaction work area specified on the transaction definition.

name4 The name of a 4-byte location to receive the size of the transaction work area, expressed as a binary value.

(Rn) A register to receive the size of the transaction work area, expressed as a binary value.

RESPONSE and REASON values for INQUIRE_TRANDEF

RESPONSE	*REASON*
OK	None
EXCEPTION	UNKNOWN_TRANSACTION_ID
INVALID	None
DISASTER	LOGIC_ERROR
PURGED	None

The INQUIRE_TRANSACTION call

The INQUIRE_TRANSACTION function is provided on the DFHXMIQX macro call. Its purpose is to allow you to obtain information about a transaction that is attached (task). In general, this call is equivalent to the EXEC CICS INQUIRE TASK command, with some minor differences.

INQUIRE_TRANSACTION

```
DFHXMIQX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_TRANSACTION),
       [TRANSACTION_TOKEN(name8),]]
       [OUT,
       [ATTACH_TIME(name8),]
       [CICS_UOW_ID(name8),]
       [DTIMEOUT(name4 | (Rn)),]
       [DYNAMIC(name1),]
       [FACILITY_NAME(name4),]
       [FACILITY_TYPE(name1),]
       [INITIAL_PROGRAM(name8),]
       [NETNAME(name8),]
       [ORIGINAL_TRANSACTION_ID(name4),]
       [OUT_TRANSACTION_TOKEN(name8),]
       [RE_ATTACHED_TRANSACTION(name1),]
       [REMOTE(name1),]
       [REMOTE_NAME(name8),]
       [REMOTE_SYSTEM(name4),]
       [RESOURCE_NAME(name8),]
       [RESOURCE_TYPE(name8),]
       [RESTART(name1),]
       [RESTART_COUNT(name2 | (Rn)),]
       [SPURGE(name1),]
       [START_CODE(name1),]
       [STATUS(name1),]
       [SUSPEND_TIME(name4 | (Rn)),]
       [SYSTEM_TRANSACTION(name1),]
       [TASK_PRIORITY(name1),]
       [TCLASS(name1),[TCLASS_NAME(name8),]]
       [TERMINATE_PROTECTED(name1),]       [TPURGE(name1),]
       [TRANNUM(name4 | string | 'string'),]
       [TRAN_PRIORITY(name1),]
       [TRAN_ROUTING_PROFILE(name8),]
       [TRANSACTION_ID(name4),]
       [USERID(name8),]
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

The descriptions of the following parameters are the same as the corresponding parameters on the INQUIRE_TRANDEF function call.

```
DTIMEOUT
DYNAMIC
INITIAL_PROGRAM
REMOTE
REMOTE_NAME
REMOTE_SYSTEM
RESTART
SPURGE
STATUS
TCLASS
TRAN_ROUTING_PROFILE
TRANSACTION_ID
```

The parameter descriptions that follow explain briefly the possible values that can be returned on an INQUIRE_TRANSACTION call. For a more detailed explanation of these parameters, see the corresponding parameter descriptions for the TRANSACTION resource definition in the *CICS Resource Definition Guide*.

ATTACH_TIME(name8)
returns the time in milliseconds since the task was attached.

name8 The name of an 8-byte location to receive the time, in packed decimal ABSTIME format.

CICS_UOW_ID(name8)
returns the CICS unit of work identifier for the task.

name8 The name of an 8-byte location to receive the unit of work id.

FACILITY_NAME(name4)
returns the name of the principal facility associated with the task.

name4 The name of a 4-byte location to receive the name of the principal facility.

FACILITY_TYPE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the type of principal facility associated with the task.

XMIQ_NONE
There is no principal facility.

XMIQ_START
The principal facility is an interval control element.

XMIQ_TD
The principal facility is a transient data queue.

XMIQ_TERMINAL
The principal facility is a terminal.

NETNAME(name8)
returns the network name of the principal facility associated with this task.

name8 The name of an 8-byte location to receive the network name.

ORIGINAL_TRANSACTION_ID(name4)
returns the transaction id that was used to attach the transaction. For example, if an alias was used at a terminal, this field returns the alias.

name4 The name of a 4-byte location to receive the name of the original transaction identifier.

OUT_TRANSACTION_TOKEN(name8)
returns the token that represents the task.

name8 The name of an 8-byte location to receive the transaction token for the task.

RE_ATTACHED_TRANSACTION(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction has been re-attached.

XMIQ_NO
The transaction has not been re-attached and the global user exit program is invoked in the same environment as the original transaction-attach.

XMIQ_YES
The transaction has been re-attached and the global user exit program is invoked in a different environment from the original transaction-attach.

RESOURCE_NAME(name8)
returns the name of a resource that the (suspended) transaction waiting for.

name8 The name of an 8-byte location to receive the name of the resource on which the transaction is waiting.

RESOURCE_TYPE(name8)
returns the type of resource that the (suspended) transaction waiting for.

name8 The name of an 8-byte location to receive the type of resource on which the transaction is waiting.

RESTART_COUNT(name2 | (Rn))
returns the number of times this instance of the transaction has been restarted.

name2 The name of a 2-byte location to receive the number of times the transaction has been restarted, expressed as a half-word binary value.

(Rn) A register to receive the number of times the transaction has been restarted, expressed as a half-word binary value.

START_CODE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating how the task was started:

C A CICS internal attach.

XMIQ_DF
The start code isn't yet known—to be set later.

XMIQ_QD
A transient data trigger level attach.

XMIQ_S
A START command without any data.

XMIQ_SD
A START command with data.

XMIQ_SZ
A front end programming interface (FEPI) attach.

XMIQ_T
A terminal input attach.

XMIQ_TT
A permanent transaction terminal attach.

SUSPEND_TIME(name4 | (Rn))
returns the length of time that the task has been in its current suspended state.

name4 The name of a 4-byte location to receive the number of seconds, rounded down, the task has been suspended, expressed as a binary value.

(Rn) A register to receive the number of seconds, rounded down, the task has been suspended, expressed as a binary value.

SYSTEM_TRANSACTION(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the task is CICS system task.

XMIQ_NO
The task is not a CICS system task.

XMIQ_YES
The task is a CICS system task.

TASK_PRIORITY(name1)
returns the combined task priority, which is the sum of the priorities defined for the terminal, transaction, and operator.

name1 The name of a 1-byte location to receive the task priority, expressed as a binary number.

TERMINATE_PROTECTED(name1)
returns, in a 1-byte location (name1), an equated value indicating whether the transaction can be killed.

XMIQ_NO
The transaction can be killed.

XMIQ_YES
The transaction cannot be killed.

TRANNUM(name4)
returns the task number of the transaction.

name4 The name of a 4-byte location to receive the task number.

TRANSACTION_TOKEN(name8)
specifies the transaction token for the task being inquired upon. This parameter is optional, and if omitted, the current task is assumed.

If you issue this call within an XXMATT global user exit program, the current task may be a CICS system task. To inquire on the user task for which XXMATT is invoked, you must specify the transaction token passed on the XXMATT exit-specific parameter list.

name8 The name of an 8-byte location that contains the transaction token.

USERID(name8)
returns the userid associated with this task.

name8 The name of an 8-byte location to receive the userid.

RESPONSE and REASON values for INQUIRE_TRANSACTION

RESPONSE	*REASON*
OK	None
DISASTER	ABEND
	LOOP
INVALID	None
EXCEPTION	NO_TRANSACTION_ENVIRONMENT
	BUFFER_TOO_SMALL
	INVALID_TRANSACTION_TOKEN
KERNERROR	None

The SET_TRANSACTION call

The SET_TRANSACTION function is provided on the DFHXMIQX macro call. Its purpose is to allow you to change the task priority and transaction class of the current task.

Note that you can use this call to change the TCLASS_NAME only when it is invoked from an XXMATT global user exit program.

SET_TRANSACTION

```
DFHXMIQX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(SET_TRANSACTION),
       [TASK_PRIORITY(name4),]
       [TCLASS_NAME(name8),]
       [TRANSACTION_TOKEN(name8),]]
       [OUT,
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

TASK_PRIORITY(name4)
specifies the new task priority being set for the task identified by TRANSACTION_TOKEN.

name4 The name of a 4-byte location that contains the new task priority number, expressed as a binary value.

TCLASS_NAME(name8)
specifies the new transaction class name that you want to associate this task with. To specify that the task is not to be in any transaction class, specify the special default system name DFHTCL00.

name8 The name of an 8-byte location that contains the name of the new transaction class. Set this field to DFHTCL00 for no transaction class.

TRANSACTION_TOKEN(name8)
specifies the transaction token that represents the task being modified. If you omit this parameter, the call defaults to the current task.

name8 The name of an 8-byte location that contains the transaction token.

RESPONSE and REASON values for SET_TRANSACTION

RESPONSE	*REASON*
OK	None
EXCEPTION	NO_TRANSACTION_ENVIRONMENT
	UNKNOWN_TCLASS
	INVALID_TRANSACTION_TOKEN
DISASTER	ABEND
	LOOP
INVALID	None
KERNERROR	None

User journaling XPI function

The XPI provides one user journaling function, which is the DFHJCJCX call WRITE_JOURNAL_DATA.

Restriction: DFHJCJCX calls cannot be used in any exit program invoked from any global user exit point in the:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain

- Transient data program.

The WRITE_JOURNAL_DATA call

WRITE_JOURNAL_DATA writes a single journal record to the journal specified in the journal model definition that matches the journal name (either a journal on an MVS system logger log stream, an SMF data set, or no record is written where DUMMY is defined in the definition).

WRITE_JOURNAL DATA

```
DFHJCJCX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(WRITE_JOURNAL_DATA),
      FROM(block-descriptor),
      JOURNALNAME(name8 | string | 'string' ) |
      JOURNAL_RECORD_ID(name2 | string | 'string'),
      WAIT(YES|NO),
      [RECORD_PREFIX(block-descriptor),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Chapter 5, "Writing initialization and shutdown programs," on page 409.

FROM(block-descriptor)
specifies the address and the length of the journal record.

The block-descriptor comprises 8 bytes of data. The first 4 bytes hold the address of the data to be written. The second 4 bytes hold the length of the data. The block-descriptor is moved by the DFHJCJCX macro call to the location JCJC_FROM, which is mapped by the DFHJCJCY DSECT.

JOURNALNAME(name8 | string | "string")
specifies the name of the CICS journal or log to which the FROM data is to be written.

JOURNAL_RECORD_ID(name2 | string | "string")
specifies a 2-character value to be written to the journal record to identify its origin.

name2 The name of a 2-byte location

string A character string that is limited to a length of 2 in the generated code

"string"
A character string enclosed in quotation marks, limited to a length of 2 in the generated code.

RECORD_PREFIX(block-descriptor)
specifies the optional user prefix.

WAIT(YES|NO)
specifies whether CICS is to wait until the record is written to the journal or log before returning control to the exit program.

RESPONSE and REASON values for WRITE_JOURNAL_DATA

RESPONSE	*REASON*
OK	None
EXCEPTION	IO_ERROR
	JOURNAL_NOT_FOUND
	JOURNAL_NOT_OPEN
	LENGTH_ERROR
	STATUS_ERROR
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "Making an XPI call" on page 382.

Appendix B. Coding entries in the z/OS Communications Server LOGON mode table

You must code your z/OS Communications Server LOGON mode table correctly for a terminal to be automatically installed.

Overview of the z/OS Communications Server LOGON mode table

CICS uses the data that you code in your z/OS Communications Server LOGON mode table when processing an automatic installation (autoinstall) request. Automatic installation functions properly only if the logmode entries that you define to z/OS Communications Server have matches among the TYPETERMs and model TERMINAL definitions that you specify to CICS.

The following tables show, for a variety of possible terminal devices, what you must code on the z/OS Communications Server MODEENT macros that define your logmode table if you want to use autoinstall. Between them they show the values that must be specified for each of the operands of the MODEENT macro. Where all bit settings of an operand's value have significance for CICS, the data is shown in hexadecimal form. If some of an operand's bit settings are not significant to CICS, its data bytes are shown as bit patterns. The bit settings that have significance for CICS are shown set to the values that CICS expects. Those bits that have no significance to CICS are shown as periods. Thus, for example:

```
01..0011
```

shows that six bits in the subject byte must be given specific values; the remaining two have no significance.

Some of the examples shown here correspond exactly to entries in the CICS-supplied LOGON mode table called ISTINCLM. Where this is so, the table gives the name of the entry in ISTINCLM.

The PSERVIC setting shows fields called aaaaaaaa, bbbbbbbb, and so on. The contents of these vary for LUTYPE0, LUTYPE2, and LUTYPE3 devices, according to how you specify certain attributes of the terminals. You can work out the values you need by looking at “PSERVIC screen size values for LUTYPEx devices” on page 880.

TYPETERM device types and pointers to related LOGON mode data

For each type of TYPETERM device, there is a reference number that has to be coded on the z/OS Communications Server MODEENT macros. You can use this information when deciding what terminals to autoinstall.

Table 48 on page 874 is a complete list of TYPETERM device types; not all of these can be used with autoinstall. Those that cannot are marked with an asterisk (*). For details about coding TYPETERM definitions, and for a list of terminals that can be autoinstalled, see the *CICS Resource Definition Guide*.

Table 48. TYPETERM device types, with cross-references to z/OS Communications Server logmode entries

TYPETERM device type	Reference number in Table 49 on page 876
DEVICE(APPC)	24
DEVICE(BCHLU)	17
DEVICE(BCHLU) SESSIONTYPE(BATCHDI)	15
DEVICE(BCHLU) SESSIONTYPE(USERPROG)	16
DEVICE(CONTLU)	10
DEVICE(INTLU)	11
DEVICE(LUTYPE2)	18
DEVICE(LUTYPE2) TERMMODEL(1)	18
DEVICE(LUTYPE3)	19
DEVICE(LUTYPE3) TERMMODEL(1)	19
DEVICE(LUTYPE4)	12
DEVICE(SCSPRINT)	11, 13
DEVICE(TLX)	8
DEVICE(TLX) SESSIONTYPE(CONTLU)	8
DEVICE(TLX) SESSIONTYPE(INTLU)	9
DEVICE(TWX)	8
DEVICE(TWX) SESSIONTYPE(CONTLU)	8
DEVICE(TWX) SESSIONTYPE(INTLU)	9
DEVICE(3270)	2
DEVICE(3270) BRACKET(NO)	1
DEVICE(3270) TERMMODEL(1)	2
DEVICE(3270) TERMMODEL(1) BRACKET(NO)	1
DEVICE(3270P)	2
DEVICE(3270P) BRACKET(NO)	1
DEVICE(3270P) TERMMODEL(1)	2
DEVICE(3270P) TERMMODEL(1) BRACKET(NO)	1
DEVICE(3275)	2
DEVICE(3275) BRACKET(NO)	1
DEVICE(3275) TERMMODEL(1)	2
DEVICE(3275) TERMMODEL(1) BRACKET(NO)	1
DEVICE(3600)	16, 22, 23
DEVICE(3600) SESSIONTYPE(PIPELINE) *	21
DEVICE(3600) SESSIONTYPE(PIPELN) *	21
DEVICE(3614) *	3
DEVICE(3650) SESSIONTYPE(PIPELINE) *	21
DEVICE(3650) SESSIONTYPE(PIPELN) *	21
DEVICE(3650) SESSIONTYPE(USERPROG) BRACKET(YES)	6

Table 48. TYPETERM device types, with cross-references to z/OS Communications Server logmode entries (continued)

TYPETERM device type	Reference number in Table 49 on page 876
DEVICE(3650) SESSIONTYPE(USERPROG) BRACKET(NO)	7
DEVICE(3650) SESSIONTYPE(3270)	5
DEVICE(3650) SESSIONTYPE(3270) BRACKET(NO)	4
DEVICE(3650) SESSIONTYPE(3653)	5
DEVICE(3650) SESSIONTYPE(3653) BRACKET(NO)	4
DEVICE(3767)	11
DEVICE(3767C)	10
DEVICE(3767I)	11
DEVICE(3770)	17
DEVICE(3770) SESSIONTYPE(BATCHDI)	15
DEVICE(3770) SESSIONTYPE(USERPROG)	16
DEVICE(3770B)	17
DEVICE(3770B) SESSIONTYPE(BATCHDI)	15
DEVICE(3770B) SESSIONTYPE(USERPROG)	16
DEVICE(3770C)	10
DEVICE(3770I)	11
DEVICE(3790)	20
DEVICE(3790) SESSIONTYPE(BATCHDI)	14
DEVICE(3790) SESSIONTYPE(SCSPRT)	13
DEVICE(3790) SESSIONTYPE(SCSPRINT)	13
DEVICE(3790) SESSIONTYPE(USERPROG)	16
DEVICE(3790) SESSIONTYPE(3277CM)	18
DEVICE(3790) SESSIONTYPE(3284CM)	19
DEVICE(3790) SESSIONTYPE(3286CM)	19

z/OS Communications Server MODEENT macro operands

The z/OS Communications Server LOGON mode table lists the MODEENT macro entries that are required for related CICS TYPETERM resources.

Look down the left side of the table for the reference number (RN) that brought you here from Table 48 on page 874. When you find it, look across to the middle column. This shows the macro operands that affect the way CICS handles automatic installation. Your MODEENT macro entries for devices to be installed must match what is specified there. Any MODEENT macro entries not shown in the table, such as PSERVIC for some reference numbers, are not tested by CICS. Any bit settings that do not matter to CICS during bind analysis for autoinstalled terminals appear as periods (.).

Note: Some fields in the PSERVIC data for LUTYPE0, LUTYPE2, and LUTYPE3 devices have values that depend on the ALTSCREEN and DEFSCREEN

characteristics of the device. For this reason, consult “PSERVIC screen size values for LUTYPEx devices” on page 880 to find out the values you need to specify instead of aaaaaaaa, bbbbbbbb, cccccccc, dddddddd, and eeeeeeee.

The right column in the table names entries in the supplied LOGON mode table, that might meet your needs. The supplied table is called ISTINCLM.

Table 49. LOGON mode table and ISTINCLM entries

RN	z/OS Communications Server MODEENT macro entries that are needed for related CICS TYPETERM definitions	Suitable supplied entries
1	FMPROF=X'02' TSPROF=X'02' PRIPROT=X'70' SECPROT=X'40' COMPROT=B'0000.000 00000.00'	
2	FMPROF=X'02' TSPROF=X'02' PRIPROT=X'71' SECPROT=X'40' COMPROT=B'0010.000 00000.00'	DSILGMOD D4B32781 D4B32782 D4B32783 D4B32784 D4B32785 NSX32702 S3270
3	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'B0' SECPROT=X'B0' COMPROT=B'0000.000 00000.00'	
4	FMPROF=X'04' TSPROF=X'03' PRIPROT=X'B0' SECPROT=X'90' COMPROT=B'0100.000 00000.00'	
5	FMPROF=X'04' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0110.000 00000.00'	
6	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'31' SECPROT=X'30' COMPROT=B'0110.000 00000.00'	INTRUSER
7	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'B0' SECPROT=X'30' COMPROT=B'0100.000 00000.00'	

Table 49. LOGON mode table and ISTINCLM entries (continued)

RN	z/OS Communications Server MODEENT macro entries that are needed for related CICS TYPETERM definitions	Suitable supplied entries
8	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0011.000 01000.00' PSERVIC=B'00000001 00000000 00000000 0000000. . . . . . 00000000 00000000 00000000 00000000 . . . . . 00000000 00000000'	
9	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0011.000 10000.00' PSERVIC=B'00000001 00000000 00000000 0000000. . . . . . 00000000 00000000 00000000 00000000 . . . . . 00000000 00000000'	SCS
10	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0011.000 01000.00' PSERVIC=X'01'	
11	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0011.000 10000.00' PSERVIC=X'01'	SCS See note 2
12	FMPROF=X'07' TSPROF=X'07' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0101.000 10000.01' PSERVIC=B'00000100 10101000 01000000 10100000 . . . . . 10101000 01000000 10100000 00000000 . . . . . 00001100 00000000'	
13	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0011.000 10000.00' PSERVIC=X'01'	SCS3790 See note 2
14	FMPROF=X'03' TSPROF=X'04' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0111.000 10000.00' PSERVIC=B'00000001 00110001 00011000 0100000. . . . . . 00000000 10010010 00000000 00000000 . . . . . 00000000 01010000'	

Table 49. LOGON mode table and ISTINCLM entries (continued)

RN	z/OS Communications Server MODEENT macro entries that are needed for related CICS TYPETERM definitions	Suitable supplied entries
15	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0111.000 10000.00' PSERVIC=B'00000001 00110001 00001100 0111000. . ..... 00000000 11010010 00000000 00000000 ..... 00000000 11010000'	
16	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0111.000 10000.00'	See note 3
17	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0111.000 10000.00' PSERVIC=B'00000001 00100000 00000000 0000000. . ..... 00000000 11000010 00000000 00000000 ..... 00000000 11000000'	
18	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=B'10..0000' COMPROT=B'0011.000 10000.00' PSERVIC=B'00000010 10000000 00000000 00000000 00000000 00000000 aaaaaaaa bbbbbbbb cccccccc dddddddd eeeeeeee'	D329001 D4A32771 D4A32772 D4A32781 D4A32782 D4A32783 D4A32784 D4A32785 D4C32771 D4C32772 D4C32781 D4C32782 D4C32783 D4C32784 D4C32785 D6327801 D6327802 D6327803 D6327804 D6327805 EMUDPCX EMU3790 SNX32702 See note 1

Table 49. LOGON mode table and ISTINCLM entries (continued)

RN	z/OS Communications Server MODEENT macro entries that are needed for related CICS TYPETERM definitions	Suitable supplied entries
19	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=B'10..0000' COMPROT=B'0011.000 10000.00' PSERVIC=B'00000011 10000000 00000000 00000000 00000000 00000000 aaaaaaaa bbbbbbbb cccccccc dddddddd eeeeeeee'	BLK3790 DSC2K DSC4K D6328902 D6328904 See note 1
20	FMPROF=X'04' TSPROF=X'03' PRIPROT=X'31' SECPROT=X'B0' COMPROT=B'0111.000'	
21	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'50' SECPROT=X'10' COMPROT=B'0000.000 00000.00'	
22	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'B0' SECPROT=X'B0' COMPROT=B'0100.000 00000.00'	IBMS3650
23	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0111.000 00000.00'	
24	TYPE=X'00' FMPROF=X'13' TSPROF=X'07' PRIPROT=X'B0' SECPROT=X'B0' COMPROT=B'.101.000 10110.01' PSERVIC=B'00000110 00000010 ..... 00000000 00000000 00000000 ..... 00000000 00000000 00000000 ..... 0010..00'	

Notes:

1. PSERVIC (RN 18 and 19): BYTE 2 BIT 0 must be set on where extended data stream (EXTDS) support is required.
2. RN 11 or 13 is used to determine the MODEENT macro operands for device SCSPRINT. However, if you have specified any of the attributes EXTENDEDDS, COLOR, PROGSYMBOLS, HILIGHT, SOSI, OUTLINE, QUERY(COLD), or QUERY(ALL) for the TYPETERM, then the COMPROT parameter of RN 13 must be modified to read COMPROT=B'0111.000 10000.00'.
3. This LOGMODE can be used for either device type 4700 in half duplex mode or device types BCHLU, 3770, 3770B and 3790 with SESSIONTYPE(USERPROG). To enable these devices to be autoinstalled with the correct model, the model names list supplied to the autoinstall exit will list

the names of models defined as DEVICE(3600) after the names of all other eligible models. The exit can be coded to select a name from the end of the list for a 4700 half duplex device.

PSERVIC screen size values for LUTYPEx devices

You can use the autoinstall model device definition options table to help you decide what screen size values you must specify on the PSERVIC operand of the z/OS Communications Server MODEENT macro, for LUTYPE0, LUTYPE2, and LUTYPE3 devices.

If, on your CICS TYPETERM definition, you code the values shown in columns 1 through 4 of Table 50, the screen size values in the CICS model bind image are as shown in column 5. The values you code for screen sizes on the PSERVIC operand must match this.

CICS treats some differently coded PSERVIC screen size specifications as equivalent. See Table 51.

Table 50. Autoinstall model device definition options

Device-type	DEFSCRN	ALTSCRN	QUERY	MODEL BIND
0,2,3	00,00	?	?	INVALID
0,2,3	12,40	,	?	0000000001
0,2,3	12,40	00,00	?	0C2800007E
0,2,3	12,40	YY,YY	?	0C28YYYY7F
0,2,3	24,80	,	NO	0000000002
3	24,80	,	COLD/ALL	0000000002
0,2	24,80	,	COLD/ALL	0000000003
0,2,3	24,80	00,00	?	185000007E
0,2,3	24,80	YY,YY	?	1850YYYY7F
0,2,3	XX,XX	,	?	XXXX00007E
0,2,3	XX,XX	00,00	?	XXXX00007E
0,2,3	XX,XX	YY,YY	?	XXXXYYYY7F

Where:

0 indicates local non-SNA 3270

2 indicates LUTYPE2

3 indicates LUTYPE3

, indicates the default

XX,XX indicates a screen size that is not 12,40 or 24,80

YY,YY indicates a screen size that is not 00,00 or blanks

? means any (that is, QUERY=ALL|COLD|NO, and ALTSCRN=any)

Table 51. Equivalent PSERVIC screen size values

Bytes 20 - 24 of CICS model bind	Valid screen size values on PSERVIC definition
0000 0000 01	0000 0000 00 0000 0000 01 0C28 0000 7E

Table 51. Equivalent PSERVIC screen size values (continued)

Bytes 20 - 24 of CICS model bind	Valid screen size values on PSERVIC definition
0000 0000 02	0000 0000 00 0000 0000 02 1850 0000 7E
0000 0000 03	0000 0000 00 0000 0000 03 1850 0000 03
xxxx 0000 7E Plus, if xxxx=1850	0000 0000 00 xxxx 0000 7E 0000 0000 02
xxxx yyyy 7F	0000 0000 00 xxxx yyyy 7F

Where:

xxxx indicates 2 bytes containing the default screen size, in hexadecimal

yyyy indicates 2 bytes containing the alternate screen size, in hexadecimal

Matching models and LOGON mode entries

This section contains a set of z/OS Communications Server LOGON mode table definitions, and their matching CICS autoinstall definitions. Each entry consists of a z/OS Communications Server logmode definition, the matching CICS TYPETERM and model TERMINAL definitions, and (for information) the BIND that CICS sends based on the specified model definition.

Note that the CICS-specific attributes are purely arbitrary. Only device attributes affect the match algorithm. It is the responsibility of the autoinstall user program to distinguish between matching models.

```
*****************************************************************
1) LOCAL NON-SNA 3277 / 3278 / 3279 (without special features)
*****************************************************************
MT32772  MODEENT LOGMODE=MT32772,  3277/8       MODEL 2
                 TYPE=1,
                 FMPROF=X'02',
                 TSPROF=X'02',
                 PRIPROT=X'71',
                 SECPROT=X'40',
                 COMPROT=X'2000',
                 PSERVIC=X'000000000000000000000200'
         OR
                 PSERVIC=X'00000000000018502B507F00' Others
         OR
                 PSERVIC=X'000000000000185000007E00' Model 2, no Altscreen
TERMINAL definition
*************************
AUTINSTNAME   ==> M3278A
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3278
INSERVICE     ==> YES
TYPETERM definition
*************************
TYPETERM      ==> T3278
GROUP         ==> PDATD
DEVICE        ==> 3270
TERMMODEL     ==> 2
LIGHTPEN      ==> YES
AUDIBLEALARM  ==> YES
UCTRAN        ==> YES
IOAREALEN     ==> 2000,2000
ERRLASTLINE   ==> YES
ERRINTENSIFY  ==> YES
```

```
USERAREALEN   ==> 32
ATI           ==> YES
TTI           ==> YES
AUTOCONNECT   ==> NO
LOGONMSG      ==> YES

BIND SENT BY CICS depends on PSERVIC value on LOGMODE definition above:
EITHER            :    01020271 40200000 00000080 00000000
                       00000000 00000002 00009300 00300000
OR                :    01020271 40200000 00000080 00000000
                       00000018 502B507F 00009300 00300000
OR                :    01020271 40200000 00000080 00000000
Real Model 2           00000018 5000007E 00009300 00300000

*****************************************************************
2) LOCAL SNA 3277/78/79 (without special features) LUTYPE2
*****************************************************************
S32782   MODEENT LOGMODE=S32782,   SNA LUTYPE2 3270
               TYPE=1,
               FMPROF=X'03',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'B0',
               COMPROT=X'3080',
               RUSIZES=X'8585',
               PSERVIC=X'028000000000185018507F00'

TERMINAL definition
*************************
AUTINSTNAME   ==> M32782
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T32782
INSERVICE     ==> YES

TYPETERM definition
*************************
TYPETERM      ==> T32782
GROUP         ==> PDATD
DEVICE        ==> LUTYPE2
TERMMODEL     ==> 2
LIGHTPEN      ==> YES
AUDIBLEALARM  ==> YES
UCTRAN        ==> YES
IOAREALEN     ==> 256,256
ERRLASTLINE   ==> YES
ERRINTENSIFY  ==> YES
USERAREALEN   ==> 32
ATI           ==> YES
TTI           ==> YES
LOGONMSG      ==> YES
DISCREQ       ==> YES
RECEIVESIZE   ==> 256
BUILDCHAIN    ==> YES

BIND SENT BY CICS :          010303B1 B0308000 0085C780 00028000
                             00000018 5018507F 00000000 00000000

*****************************************************************
3) 3770 BATCH LU (3777)
*****************************************************************
BATCH    MODEENT LOGMODE=BATCH,    3770 BATCH
               TYPE=1,
               FMPROF=X'03',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'B0',
               COMPROT=X'7080',
               PSERVIC=X'01310C70E100D20000E100D0'
```

```
TERMINAL definition
*************************
AUTINSTNAME   ==> M3770
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3770
INSERVICE     ==> YES

TYPETERM definition
*************************
TYPETERM      ==> T3770
GROUP         ==> PDATD
DEVICE        ==> 3770
SESSIONTYPE   ==> BATCHDI
PAGESIZE      ==> 12,80
DISCREQ       ==> YES
AUTOPAGE      ==> YES
RECEIVESIZE   ==> 256
SENDSIZE      ==> 256
IOAREALEN     ==> 256,2048
BUILDCHAIN    ==> YES
BRACKET       ==> YES
ATI           ==> YES
TTI           ==> YES
AUTOCONNECT   ==> NO
HORIZFORM     ==> YES
VERTFORM      ==> YES
LDCLIST       ==> LDC2
Needs LDC declaration in TCT :
LDC2   DFHTCT TYPE=LDC,LOCAL=INITIAL
       DFHTCT TYPE=LDC,LDC=BCHLU
       DFHTCT TYPE=LDC,LOCAL=FINAL

BIND SENT BY CICS :          010303B1 B0708000 00000080 0001310C
                             70E100D2 0000E100 D0000000 00000000

*********************************************************************
4) 6670 LUTYPE4
*********************************************************************
S6670    MODEENT LOGMODE=S6670,    6670 LUTYPE4
                 TYPE=1,
                 FMPROF=X'07',
                 TSPROF=X'07',
                 RUSIZES=X'8585',
                 PRIPROT=X'B1',
                 SECPROT=X'B0',
                 COMPROT=X'5081',
                 PSERVIC=X'04A840A000A840A000000C00'

TERMINAL definition
*************************
AUTINSTNAME   ==> M6670
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T6670
INSERVICE     ==> YES

TYPETERM definition
*************************
TYPETERM      ==> T6670
GROUP         ==> PDATD
DEVICE        ==> LUTYPE4
BUILDCHAIN    ==> YES
DISCREQ       ==> YES
RECEIVESIZE   ==> 256
UCTRAN        ==> YES
IOAREALEN     ==> 256,4096
FORMFEED      ==> YES
HORIZFORM     ==> YES
VERTFORM      ==> YES
```

```
ATI           ==> YES
TTI           ==> YES
PAGESIZE      ==> 50,80
AUTOPAGE      ==> YES
LOGONMSG      ==> NO
LDCLIST       ==> LDC1

Needs LDC declaration in TCT :
LDCS     DFHTCT TYPE=LDC,LDC=SYSTEM
LDC1     DFHTCT TYPE=LDC,LOCAL=INITIAL
         DFHTCT TYPE=LDC,DVC=(BLUCON,01),PROFILE=DEFAULT,LDC=PC,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(BLUPRT,02),PROFILE=BASE,LDC=PP,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(BLUPRT,08),PROFILE=BASE,LDC=P8,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(BLUPRT,08),PROFILE=DEFAULT,LDC=DP,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(BLUPCH,03),PROFILE=JOB,LDC=PM,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(BLUPCH,03),PROFILE=DEFAULT,LDC=DM,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(WPMED1,04),PROFILE=WPRAW,LDC=P1,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(WPMED1,04),PROFILE=DEFAULT,LDC=D1,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(WPMED2,05),PROFILE=OII1,LDC=P2,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(WPMED2,05),PROFILE=DEFAULT,LDC=D2,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(WPMED3,06),PROFILE=OII2,LDC=P3,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,DVC=(WPMED4,07),PROFILE=OII3,LDC=P4,
               PGESIZE=(50,80),PGESTAT=AUTOPAGE
         DFHTCT TYPE=LDC,LOCAL=FINAL

BIND SENT BY CICS :          010707B1 B0508100 00858580 0004A840
                             A000A840 A000000C 00000000 00000000

*****************************************************************
5) 3790 FULL FUNCTION LU
*****************************************************************
S3790A   MODEENT LOGMODE=S3790A,   3790 FULL FUNCTION LU
               TYPE=1,
               FMPROF=X'04',
               TSPROF=X'04',
               PRIPROT=X'B1',
               SECPROT=X'B0',
               RUSIZES=X'8585',
               COMPROT=X'7080'

TERMINAL definition
*************************
AUTINSTNAME   ==> M3790A
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3790A
INSERVICE     ==> YES

TYPETERM definition
*************************
TYPETERM      ==> T3790A
GROUP         ==> PDATD
DEVICE        ==> 3790
SENDSIZE      ==> 256
RECEIVESIZE   ==> 256
SESSIONTYPE   ==> USERPROG
```

```
BRACKET       ==> YES
IOAREALEN     ==> 256
ATI           ==> YES
TTI           ==> YES

BIND SENT BY CICS :          010404B1 B0708000 00858580 00000000

*****************************************************************
6) 3790 BATCH DATA INTERCHANGE
*****************************************************************
S3790B   MODEENT LOGMODE=S3790B,   3790 BATCH
               TYPE=1,
               FMPROF=X'03',
               TSPROF=X'04',
               PRIPROT=X'B1',
               SECPROT=X'B0',
               COMPROT=X'7080',
               RUSIZES=X'8585',
               PSERVIC=X'013118400000920000E10050'

TERMINAL definition
*************************
AUTINSTNAME   ==> M3790B
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3790B
INSERVICE     ==> YES
TERMPRIORITY  ==> 50

TYPETERM definition
*************************
TYPETERM      ==> T3790B
GROUP         ==> PDATD
DEVICE        ==> 3790
SESSIONTYPE   ==> BATCHDI
AUTOPAGE      ==> YES
BUILDCHAIN    ==> YES
OBOPERID      ==> YES
IOAREALEN     ==> 256,2048
RELREQ        ==> YES
SENDSIZE      ==> 256
RECEIVESIZE   ==> 256
ATI           ==> YES
TTI           ==> YES
LDCLIST       ==> LDC2

Needs LDC declaration in TCT :
LDC2   DFHTCT TYPE=LDC,LOCAL=INITIAL
       DFHTCT TYPE=LDC,LDC=BCHLU
       DFHTCT TYPE=LDC,LOCAL=FINAL

BIND SENT BY CICS :          010304B1 B0708000 00858580 00013118
                             40000092 0000E100 50000000 00000000

*****************************************************************
7) 3790 SCSPRT
*****************************************************************
S3790C   MODEENT LOGMODE=S3790C,   3790 WITH SCS
               TYPE=1,
               FMPROF=X'03',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'B0',
               COMPROT=X'3080',
               RUSIZES=X'8585',
               PSERVIC=X'010000000000000000000000'

TERMINAL definition
*************************
AUTINSTNAME   ==> M3790C
```

```
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3790C
INSERVICE     ==> YES

TYPETERM definition
************************
TYPETERM      ==> T3790C    Note that CEDA changes DEVICE=3790,
GROUP         ==> PDATD     SESSIONTYPE=SCSPRT to DEVICE=SCSPRINT,
DEVICE        ==> 3790      SESSIONTYPE=blanks, PRINTERTYPE=3284.
SESSIONTYPE   ==> SCSPRT
BRACKET       ==> YES
SENDSIZE      ==> 256
RECEIVESIZE   ==> 256
ATI           ==> YES
TTI           ==> YES

BIND SENT BY CICS :          010303B1 B0308000 00858580 00010000

*********************************************************************
8) 3767 INTERACTIVE (FLIP-FLOP) LU
*********************************************************************
S3767    MODEENT LOGMODE=S3767,    3767 INTERACTIVE
                 TYPE=1,
                 FMPROF=X'03',
                 TSPROF=X'03',
                 PRIPROT=X'B1',
                 SECPROT=X'90',
                 COMPROT=X'3080',
                 PSERVIC=X'010000000000000000000000'

TERMINAL definition
************************
AUTINSTNAME   ==> M3767
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TERMPRIORITY  ==> 60
TYPETERM      ==> T3767
INSERVICE     ==> YES

TYPETERM definition
************************
TYPETERM      ==> T3767
GROUP         ==> PDATD
DEVICE        ==> 3767
VERTFORM      ==> YES
HORIZFORM     ==> YES
RELREQ        ==> YES
DISCREQ       ==> YES
IOAREALEN     ==> 256
AUTOPAGE      ==> NO
PAGESIZE      ==> 12,80
ATI           ==> YES
TTI           ==> YES
BRACKET       ==> YES
RECEIVESIZE   ==> 256
SENDSIZE      ==> 256

BIND SENT BY CICS :          010303B1 90308000 00000080 00010000

*********************************************************************
9) 3650 INTERPRETER LU
     (SESTYPE = USERPROG  BRACKET = YES)
*********************************************************************
S3650A   MODEENT LOGMODE=S3650A,   3650 SESTYPE=USERPROG
                 TYPE=1,           BRACKET=YES
                 FMPROF=X'04',
                 TSPROF=X'04',
                 PRIPROT=X'31',
                 SECPROT=X'30',
                 COMPROT=X'6000'
```

```
TERMINAL definition
*************************
AUTINSTNAME   ==> M3650A
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3650A
INSERVICE     ==> YES

TYPETERM definition
*************************
TYPETERM      ==> T3650A
GROUP         ==> PDATD
DEVICE        ==> 3650
SESSIONTYPE   ==> USERPROG
ROUTEDMSGS    ==> SPECIFIC
FMHPARM       ==> YES
RELREQ        ==> YES
DISCREQ       ==> YES
BRACKET       ==> YES
RECEIVESIZE   ==> 256
IOAREALEN     ==> 256,256
ATI           ==> YES
TTI           ==> YES
AUTOCONNECT   ==> NO

BIND SENT BY CICS :          01040431 30600000 00000080 00000000

*********************************************************************
10) 3650 HOST CONVERSATIONAL (3270) LU
*********************************************************************
S3650B   MODEENT LOGMODE=S3650B,   3650 SESTYPE=3270
               TYPE=1,             AND SESTYPE=3653
               FMPROF=X'04',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'90',
               COMPROT=X'6000'

TERMINAL definition
*************************
AUTINSTNAME   ==> M3650B1
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3650B1
INSERVICE     ==> YES

TYPETERM definition
*************************
TYPETERM      ==> T3650B1
GROUP         ==> PDATD
DEVICE        ==> 3650
OBFORMAT      ==> YES
SESSIONTYPE   ==> 3270
RELREQ        ==> YES
DISCREQ       ==> YES
IOAREALEN     ==> 256
BRACKET       ==> YES
RECEIVESIZE   ==> 240
ATI           ==> NO
TTI           ==> YES

BIND SENT BY CICS :          010403B1 90600000 00000080 00000000

*********************************************************************
11) 3650 HOST CONVERSATIONAL (3653) LU
     (N.B. LOGMODE SAME AS HC (3270) LU)
*********************************************************************
S3650B   MODEENT LOGMODE=S3650B,   3650 SESTYPE=3270
               TYPE=1,             AND SESTYPE=3653
               FMPROF=X'04',
```

```
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'90',
               COMPROT=X'6000'

TERMINAL definition
*************************
AUTINSTNAME   ==> M3650B2
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3650B2
INSERVICE     ==> YES

TYPETERM definition
*************************
TYPETERM      ==> T3650B2
GROUP         ==> PDATD
DEVICE        ==> 3650
SESSIONTYPE   ==> 3653
RELREQ        ==> YES
DISCREQ       ==> NO
BRACKET       ==> YES
IOAREALEN     ==> 256
RECEIVESIZE   ==> 240
ROUTEDMSGS    ==> NONE
ATI           ==> NO
TTI           ==> YES

BIND SENT BY CICS :          010403B1 90600000 00000080 00000000

*********************************************************************
12) 3650 HOST COMMAND PROCESSOR LU
     (SESTYPE = USERPROG  BRACKET = NO)
*********************************************************************
S3650C   MODEENT LOGMODE=S3650C,   3650 SESTYPE=USERPROG
               TYPE=1,             BRACKET=NO
               FMPROF=X'04',
               TSPROF=X'04',
               PRIPROT=X'B0',
               SECPROT=X'30',
               COMPROT=X'4000'

TERMINAL definition
*************************
AUTINSTNAME   ==> M3650C
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3650C
INSERVICE     ==> YES

TYPETERM definition
*************************
TYPETERM      ==> T3650C
GROUP         ==> PDATD
DEVICE        ==> 3650
SESSIONTYPE   ==> USERPROG
BRACKET       ==> NO
RELREQ        ==> NO
DISCREQ       ==> NO
RECEIVESIZE   ==> 256
IOAREALEN     ==> 256
ATI           ==> YES
TTI           ==> YES

BIND SENT BY CICS :          01040430 30400000 00000080 00000000

*********************************************************************
13) 8815 SCANMASTER (APPC SINGLE SESSION)
*********************************************************************
SIN62  MODEENT LOGMODE=SIN62,      8815 SCANMASTER.
               TYPE=0,
               FMPROF=X'13',
```

```
               TSPROF=X'07',
               PRIPROT=X'B0',
               SECPROT=X'B0',
               COMPROT=X'50B1',
               PSNDPAC=X'00',
               SRCVPAC=X'00',
               SSNDPAC=X'00',
               RUSIZES=X'8585',
               PSERVIC=X'060200000000000000002C00'

TERMINAL definition
************************
AUTINSTNAME  ==> MLU62
AUTINSTMODEL ==> ONLY
GROUP        ==> PDATD
TYPETERM     ==> SINLU62
INSERVICE    ==> YES

TYPETERM definition
************************
TYPETERM     ==> SINLU62
GROUP        ==> PDATD
DEVICE       ==> APPC
RECEIVESIZE  ==> 2048
SENDSIZE     ==> 2048
ATI          ==> YES
TTI          ==> YES
Note: There is no RDO keyword equivalent of the MACRO
keyword 'FEATURE=SINGLE', because this is assumed with
RDO DEFINE TYPETERM when DEVICE=APPC.

BIND SENT BY CICS :          001307B0 B050B100 00858580 00060200
                             00000000 0000002C 00000800 00000000
                             0000001D 00090240 40404040 40404009
                             03006765 71D98A6C 300704C3 C9C3E2E6
                             F1000000 00000000 00000000 00000000

*********************************************************************
14) 3290 (SDLC)
*********************************************************************
S3290    MODEENT LOGMODE=S3290,    3290 SDLC
               TYPE=1,
               FMPROF=X'03',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'90',
               COMPROT=X'3080',
               RUSIZES=X'8787',
               PSERVIC=X'02800000000018503EA07F00'

TERMINAL definition
************************
AUTINSTNAME   ==> M3290
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3290
INSERVICE     ==> YES

TYPETERM definition
************************
TYPETERM      ==> T3290
GROUP         ==> PDATD
DEVICE        ==> LUTYPE2
TERMMODEL     ==> 2
ALTSCREEN     ==> 62,160
DEFSCREEN     ==> 24,80
AUDIBLEALARM  ==> YES
UCTRAN        ==> YES
IOAREALEN     ==> 2000,2000
ERRLASTLINE   ==> YES
ERRINTENSIFY  ==> YES
```

```
USERAREALEN   ==> 32
ATI           ==> YES
TTI           ==> YES
LOGONMSG      ==> YES
ERRHILIGHT    ==> BLINK
RECEIVESIZE   ==> 1024

BIND SENT BY CICS :          010303B1 90308000 00878780 00028000
                             00000018 503EA07F 00000000 00000000

*********************************************************************
15) 3601 WITH A 3604 ATTACHED
*********************************************************************
S3600    MODEENT LOGMODE=S3600,    3601
                 TYPE=1,
                 FMPROF=X'04',
                 TSPROF=X'04',
                 PRIPROT=X'B1',
                 SECPROT=X'B0',
                 COMPROT=X'7000',
                 RUSIZES=X'0000'

TERMINAL definition
************************
AUTINSTNAME   ==> M3600
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TERMPRIORITY  ==> 50
TYPETERM      ==> T3600
INSERVICE     ==> YES

TYPETERM definition
************************
TYPETERM      ==> T3600
GROUP         ==> PDATD
DEVICE        ==> 3600
AUTOPAGE      ==> NO
PAGESIZE      ==> 6,40
RELREQ        ==> YES
DISCREQ       ==> NO
IOAREALEN     ==> 256
SENDSIZE      ==> 224
RECEIVESIZE   ==> 256
USERAREALEN   ==> 100
ATI           ==> NO
TTI           ==> YES
BRACKET       ==> YES
LDCLIST       ==> BMSLLDC1

Needs LDC declaration in TCT :
BMSLLDC1 DFHTCT TYPE=LDCLIST,
                LDC=(DS,JP,PB=5,LP,MS)
         DFHTCT TYPE=LDC,
                LDC=(DS=1),
                DVC=3604,
                PGESIZE=(6,40),
                PGESTAT=PAGE
         DFHTCT TYPE=LDC,LDC=SYSTEM

BIND SENT BY CICS :          010404B1 B0700000 00000080 00000000
```

LOGON mode definitions for CICS-supplied autoinstall models

This chapter contains z/OS Communications Server LOGON mode table example definitions that match the CICS-supplied TYPETERM and model TERMINAL definitions for autoinstall.

The first six entries are example definitions; that is, they are not supplied with z/OS Communications Server.

```
DFHLU3   MODEENT LOGMODE=DFHLU3,   LU TYPE 3 PRINTER.
               TYPE=1,
               FMPROF=X'03',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'B0',
               COMPROT=X'3080',
               RUSIZES=X'8585',
               PSERVIC=X'038000000000000000000200'
DFHSCSP  MODEENT LOGMODE=DFHSCSP,  LU TYPE 1 SCS PRINTER
               TYPE=1,
               FMPROF=X'03',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'B0',
               COMPROT=X'7080',
               RUSIZES=X'8585',
               PSERVIC=X'010000010000000000000000'
DFHLU62T MODEENT LOGMODE=DFHLU62T, APPC SINGLE-SESSION
               TYPE=0,
               FMPROF=X'13',
               TSPROF=X'07',
               PRIPROT=X'B0',
               SECPROT=X'B0',
               COMPROT=X'50B1',
               RUSIZES=X'8888',
               PSERVIC=X'060200000000000000002C00'
DFH3270  MODEENT LOGMODE=DFH3270,  3270
               TYPE=1,
               FMPROF=X'02',
               TSPROF=X'02',
               PRIPROT=X'71',
               SECPROT=X'40',
               COMPROT=X'2000',
               RUSIZES=X'0000'
DFH3270P MODEENT LOGMODE=DFH3270P, 3284/3286 BISYNC 3270P (QUERY)
               TYPE=1,
               FMPROF=X'02',
               TSPROF=X'02',
               PRIPROT=X'71',
               SECPROT=X'40',
               COMPROT=X'2000',
               RUSIZES=X'0000'
DFHLU2   MODEENT LOGMODE=DFHLU2,   SNA LUTYPE2 3270
               TYPE=1,
               FMPROF=X'03',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'B0',
               COMPROT=X'3080',
               RUSIZES=X'85C7',
               PSERVIC=X'028000000000000000000300'
```

The following entries are those LOGMODE definitions supplied by z/OS Communications Server that match CICS-supplied TYPETERM definitions.

```
DFHLU0E2 MODEENT LOGMODE=NSX32702,  LU0 model 2 queryable
               FMPROF=X'02',
               TSPROF=X'02',
               PRIPROT=X'71',
               SECPROT=X'40',
               COMPROT=X'2000',
               RUSIZES=X'0000',
               PSERVIC=X'008000000000185000007E00'
```

```
DFHLU0M2 MODEENT LOGMODE=D4B32782,  LU0 model 2 nonqueryable
                 FMPROF=X'02',
                 TSPROF=X'02',
                 PRIPROT=X'71',
                 SECPROT=X'40',
                 COMPROT=X'2000',
                 RUSIZES=X'0000',
                 PSERVIC=X'000000000000185000007E00'
DFHLU0M3 MODEENT LOGMODE=D4B32783,  LU0 model 3 nonqueryable
                 FMPROF=X'02',
                 TSPROF=X'02',
                 PRIPROT=X'71',
                 SECPROT=X'40',
                 COMPROT=X'2000',
                 RUSIZES=X'0000',
                 PSERVIC=X'000000000000185020507F00'
DFHLU0M4 MODEENT LOGMODE=D4B32784,  LU0 model 4 nonqueryable
                 FMPROF=X'02',
                 TSPROF=X'02',
                 PRIPROT=X'71',
                 SECPROT=X'40',
                 COMPROT=X'2000',
                 RUSIZES=X'0000',
                 PSERVIC=X'00000000000018502B507F00'
DFHLU0M5 MODEENT LOGMODE=D4B32785,  LU0 model 5 nonqueryable
                 FMPROF=X'02',
                 TSPROF=X'02',
                 PRIPROT=X'71',
                 SECPROT=X'40',
                 COMPROT=X'2000',
                 RUSIZES=X'0000',
                 PSERVIC=X'00000000000018501B847F00'
DFHLU2E2 MODEENT LOGMODE=SNX32702,  LU2 model 2 queryable
                 FMPROF=X'03',
                 TSPROF=X'03',
                 PRIPROT=X'B1',
                 SECPROT=X'90',
                 COMPROT=X'3080',
                 RUSIZES=X'87F8',
                 PSERVIC=X'028000000000185000007E00'
DFHLU2E3 MODEENT LOGMODE=SNX32703,  LU2 model 3 queryable
                 FMPROF=X'03',
                 TSPROF=X'03',
                 PRIPROT=X'B1',
                 SECPROT=X'90',
                 COMPROT=X'3080',
                 RUSIZES=X'87F8',
                 PSERVIC=X'028000000000185020507F00'
DFHLU2E4 MODEENT LOGMODE=SNX32704,  LU2 model 4 queryable
                 FMPROF=X'03',
                 TSPROF=X'03',
                 PRIPROT=X'B1',
                 SECPROT=X'90',
                 COMPROT=X'3080',
                 RUSIZES=X'87F8',
                 PSERVIC=X'02800000000018502B507F00'
DFHLU2M2 MODEENT LOGMODE=D4A32782,  LU2 model 2 nonqueryable
                 FMPROF=X'03',
                 TSPROF=X'03',
                 PRIPROT=X'B1',
                 SECPROT=X'90',
                 COMPROT=X'3080',
                 RUSIZES=X'87C7',
                 PSERVIC=X'020000000000185000007E00'
```

```
DFHLU2M3 MODEENT LOGMODE=D4A32783,  LU2 model 3 nonqueryable
               FMPROF=X'03',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'90',
               COMPROT=X'3080',
               RUSIZES=X'87C7',
               PSERVIC=X'020000000000185020507F00'
DFHLU2M4 MODEENT LOGMODE=D4A32784,  LU2 model 4 nonqueryable
               FMPROF=X'03',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'90',
               COMPROT=X'3080',
               RUSIZES=X'87C7',
               PSERVIC=X'02000000000018502B507F00'
DFHLU2M5 MODEENT LOGMODE=D4A32785,  LU2 model 5 nonqueryable
               FMPROF=X'03',
               TSPROF=X'03',
               PRIPROT=X'B1',
               SECPROT=X'90',
               COMPROT=X'3080',
               RUSIZES=X'87C7',
               PSERVIC=X'02000000000018501B847F00'
```

Appendix C. Default actions of the node abnormal condition program

The default actions of the node abnormal condition program, DFHZNAC, vary, depending on the terminal error code and system sense codes received from z/OS Communications Server.

In most cases, DFHZNAC issues messages and sets one or more "action flags" in the communication area passed to the node error program, DFHZNEP. DFHZNEP then has the opportunity to change the default actions (though not the messages) by setting or resetting flags. (Note, however, that in some circumstances, the actions taken can vary from the actions set, depending on the state of the node at the time of the error.)

For more information about DFHZNAC and DFHZNEP, see Chapter 11, "Writing a node error program," on page 465.

DFHZNAC—default actions for terminal error codes

Terminal error codes from z/OS Communications Server are put in a 1-byte field (TWAEC) of the communications area passed to DFHZNEP.

Table 52 shows the message issued and action flags set by DFHZNAC for each terminal error code.

For error codes with CICS messages associated with them, see the *CICS Messages and Codes Vol 1* manual for descriptions of the corresponding error conditions.

The figures in the "**Action flags set**" column are translated into bit settings and explained in Table 55 on page 908.

Table 52. Messages issued and flags set by DFHZNAC for specific error codes

Error code	Symbolic label	Message	Action flags set
X'10'	TCZSRCTU	DFHZC2405	18
X'11'	TCZSRCBF	DFHZC2403	2 5 7 18 24
X'13'	TCZSRCVH	DFHZC2416	7 18 24
X'14'	TCZLRCER	DFHZC2404	2 3 7 9 10 11 23 24
X'15'	TCZSRCPF	DFHZC2407	2 3 7 9 10 11 24
X'16'	TCZDMIT	DFHZC3492	None
X'18'	TCZLRCNR	DFHZC2404	2 3 7 9 10 11 23 24
X'19'	TCZSRCTS	DFHZC2406	9 10 11 18
X'1A'	TCZSRCVE	DFHZC2408	2 3 7 9 10 11 24
X'1D'	TCZSRCVI	DFHZC2417	2 7 24
X'1E'	TCZSRCV2	DFHZC2408	2 3 7 9 10 11 24
X'20'	TCZVTAMI	DFHZC2417	None
X'21'	TCZLUCF1	DFHZC4902	3 7 9 10 11 24
X'22'	TCZLUCF2	DFHZC4903	3 7 9 10 11 24

Table 52. Messages issued and flags set by DFHZNAC for specific error codes (continued)

Error code	Symbolic label	Message	Action flags set
X'23'	TCZFSMBE	DFHZC4904	3 7 9 10 11 24
X'24'	TCZFSMCS	DFHZC4905	3 7 9 10 11 24
X'25'	TCZFSMCR	DFHZC4906	3 7 9 10 11 24
X'26'	TCZSDLER	DFHZC4907	3 7 9 10 11 24
X'28'	TCZRVLER	DFHZC4909	3 7 9 10 11 24
X'29'	TCZRVLRB	DFHZC4910	3 7 9 10 11 24
X'2A'	TCZRLPEX	DFHZC4911	2 3 7 9 10 11 24
X'2B'	TCZRLPBD	DFHZC4912	2 3 7 9 10 11 24
X'2C'	TCZRLPDR	DFHZC4913	2 3 7 9 10 11 24
X'2D'	TCZRLPIL	DFHZC4914	2 3 7 9 10 11 24
X'2E'	TCZRLPEC	DFHZC4915	2 3 7 9 10 11 24
X'2F'	TCZRLPRR	DFHZC4916	2 3 7 9 10 11 24
X'30'	TCZRLPIF	DFHZC4917	2 3 7 9 10 11 24
X'31'	TCZRLPIR	DFHZC4918	2 3 7 9 10 11 24
X'32'	TCZRLXCL	DFHZC4922	7 20
X'33'	TCZIVIND	DFHZC4919	2 3 7 9 10 11 24
X'34'	TCZIVDAT	DFHZC4920	2 3 7 9 10 11 24
X'35'	TCZRTMT	DFHZC4930	2 3 7 9 10 11 24
X'36'	TCZXSBL	None	24
X'37'	TCZXSHRA	DFHZC3470	9 10 11 24
X'38'	TCZXSWAS	DFHZC6596	2 3 7 15 24
X'39'	TCZXSABN	DFHZC6595	2 3 5 7 24
X'3A'	TCZXSHR	DFHZC6594	7 24
X'3B'	TCZXSBC	DFHZC6593	None
X'3C'	TCZXUVAR	DFHZC3488	2 3 7 9 10 11 24
X'3D'	TCZXMSG	None	None
X'3E'	TCZXERR	DFHZC6591	7 9 10 11 15 24
X'3F'	TCZXRST	DFHZC6590	None
X'40'	TCZINCPY	DFHZC2489	3 9 11
X'41'	TCZTOLRQ	DFHZC2490	2 3 7 9 10 11 15 24
X'42'	TCZUNPRT	DFHZC2497 - See 1 on page 901	None
X'43'	TCZCPYNS	DFHZC2434	3 11
X'44'	TCZSRCDE	DFHZC2456	2 3 7 9 10 11 24
X'45'	TCZCHMX	DFHZC3400	3 10 11 22
X'46'	TCZOCIR	DFHZC3402	3 9 10 11
X'47'	TCZGMMS	None 2 on page 901	13
X'48'	TCZOPSIN	DFHZC3461	7 , 8
X'49'	TCZCLSIN	DFHZC3462	7
X'4A'	TCZOPACB	DFHZC3463	None

Table 52. Messages issued and flags set by DFHZNAC for specific error codes (continued)

Error code	Symbolic label	Message	Action flags set
X'4B'	TCZICPUT	DFHZC2498	None
X'4C'	TCZDSPCL	DFHZC3481	2 3 7 9 10 11 24
X'4D'	TCZSLSRL	DFHZC3473	None
X'4E'	TCZUNBFE	DFHZC3479	2 3 7 9 10 11 24
X'4F'	TCZCNOS0	None	None
X'50'	TCZSDRE3	DFHZC3417	3 7 9 10 11 24
X'51'	TCZBDPRI	DFHZC3418	3 7 9 10 11 24
X'52'	TCZBDUAC	DFHZC3419	2 3 5 7
X'53'	TCZBDTOS	DFHZC3420	7 20
X'54'	TCZUNBIS	DFHZC3434	2 3 7 9 10 11 24
X'55'	TCZEMWBK	DFHZC3440	None
X'56'	TCZXRFVS	DFHZC6598	None
X'57'	TCZRELIS	DFHZC3464	7 20
X'58'	TCZERMGR	DFHZC3433	7
X'59'	TCZROCT	DFHZC2443	2 3 7 9 10 11 24
X'5A'	TCZSBIRV	DFHZC3421	7 20
X'5B'	TCZNSP01	DFHZC3422	2 3 7 9 10 11 18 24
X'5C'	TCZNSP02	DFHZC3424	7 9 10 11 15 24
X'5D'	TCZPRDTO	DFHZC0101	None
X'5E'	TCZBRUAC	DFHZC3454	2 3 5 7 18 24
X'5F'	TCZBDSQP	DFHZC3455	2 3 5 7 18 24
X'60'	TCZUNCMD	DFHZC2421	2 3 7 9 10 11 24
X'62'	TCZVTAMQ	None 3 on page 901	24
X'63'	TCZVTAMO	DFHZC3441	None
X'64'	TCZVTAMA	DFHZC3443	None
X'65'	TCZINVRR	DFHZC2448	2 3 7 10 11 22 23 24
X'66'	TCZSIGR	DFHZC3452	None
X'67'	TCZVTAMK	DFHZC3442	None
X'69'	TCZSEXOS	DFHZC3466	7 20 23
X'6A'	TCZTIOAE	DFHZC3444	1 2 3 7 9 10 11 24
X'6B'	TCZNOTNA	DFHZC3495	7 24
X'6C'	TCZPSAF	DFHZC0155	3 6 7 9 10 11 24
X'6D'	TCZPSAR	DFHZC0156	7
X'70'	TCZCLRRV	DFHZC3468	7 9 10 11 15 24
X'71'	TCZPSLE	DFHZC0147	3 6 7 9 10 11 24
X'72'	TCZPSVF	DFHZC0148	7 9 10 11 24
X'73'	TCZSDSE4	DFHZC2437	3 9 11
X'74'	TCZSDSE5	DFHZC2423	3 7 9 10 11 24
X'75'	TCZSESE1	DFHZC2424	3 7 9 10 11 15 24
X'76'	TCZLGNA	DFHZC2487	3

Table 52. Messages issued and flags set by DFHZNAC for specific error codes (continued)

Error code	Symbolic label	Message	Action flags set
X'77'	TCZDMRY	DFHZC2488	None
X'78'	TCZSDRE2	DFHZC2430	3 9 11 22
X'79'	TCZPSRAF	DFHZC0145	3 6 7 9 10 11 24
X'7A'	TCZPSRAC	DFHZC0144	7 11
X'7C'	TCZPSANR	DFHZC0157	3 7 9 10 11 24
X'7D'	TCZRABUS	DFHZC4949	2 3 7 9 10 11 24
X'80'	TCZSRCSP	DFHZC2414	None
X'81'	TCZSSXNR	DFHZC2432	None
X'82'	TCZSSXUC	DFHZC2419	2 3 7 9 10 11 23 24
X'83'	TCZSSXAR	DFHZC2450	None
X'84'	TCZSSXIB	DFHZC2446	2 3 7 9 10 11 23 24
X'85'	TCZUNEGR	DFHZC3409	2 3 7 9 10 11 23 24
X'88'	TCZLEXCI	DFHZC2467	2 3 7 9 10 11 23 24
X'89'	TCZLEXUS	DFHZC2468	2 3 7 9 10 11 24
X'8A'	TCZLUSRR	DFHZC4937	2 3 5 7 24
X'8B'	TCZLUSRF	DFHZC4938	2 3 5 7 24
X'8C'	TCZLUPUN	DFHZC4939	2 3 5 7 24
X'8D'	TCZLUPLK	DFHZC4941	2 3 5 7 24
X'8E'	TCZLUPEX	DFHZC4942	2 3 5 7 24
X'8F'	TCZLUSKN	DFHZC4940	2 3 5 7 24
X'90'	TCZLGCER	DFHZC2422	1 2 3 6 9 10 11 23 24
X'91'	TCZRSTLE	DFHZC2429	3 10 11
X'92'	TCZSDSE6	DFHZC2428	3 9 11
X'93'	TCZRACET	DFHZC2455	2 3 9 10 11
X'94'	TCZRACES	DFHZC2426	2 3 9 10 11 22
X'95'	TCZSDSE8	DFHZC2445	3 9 11
X'96'	TCZRVSZ1	DFHZC2435	3 7 10 11 24
X'97'	TCZRVSZ3	DFHZC2436	3 10 11
X'98'	TCZACT01	DFHZC2439	2 18
X'99'	TCZSDSE7	DFHZC2459	3 9 11
X'9A'	TCZDOMCF	DFHZC2447	3 9 10 11 23
X'9B'	TCZRACNL	DFHZC2486	3
X'9D'	TCZRSPER	DFHZC3465	1 2 3 7 9 10 11 23
X'9E'	TCZDEVND	DFHZC3472	None
X'A0'	TCZNOISC	DFHZC3480	7 23 24
X'A1'	TCZRVSZ2	DFHZC2438	3 10 11
X'A2'	TCZPRGE	DFHZC4945	3 7 9 10 11 24
X'A3'	TCZBKTSE	DFHZC2444	2 3 7 9 10 11 24
X'A7'	TCZBOEB	DFHZC2449	2 3 7 11 18 22 24
X'A8'	TCZFMHLE	DFHZC2471	2 3 4 7 10 11 22 24

Table 52. Messages issued and flags set by DFHZNAC for specific error codes (continued)

Error code	Symbolic label	Message	Action flags set
X'A9'	TCZRACRF	DFHZC2472	11
X'AA'	TCZSDSE9	DFHZC2473	3 9 11
X'AB'	TCZLUERR	DFHZC3470	7 9 10 11 24
X'AC'	TCZVRDAC	DFHZC3474	7 9 10 11 24
X'AD'	TCZNRLUF	DFHZC3475	7 9 10 11 24
X'AE'	TCZRCLUF	DFHZC3476	7 9 10 11 24
X'AF'	TCZCLEAN	DFHZC3477	7 9 10 11 24
X'B0'	TCZEXRO	DFHZC3491	7 15 24
X'B1'	TCZRPLAC	DFHZC2401	2 3 7 9 10 11 23 24
X'B2'	TCZSDAUC	DFHZC2425	3 7 9 10 11 15 24
X'B3'	TCZBDBND	DFHZC4929	2 3 5 7 24
X'B4'	TCZRSNE	DFHZC2402	3 11
X'B5'	TCZSAXUC	DFHZC2420	2 3 7 9 10 11 23 24
X'B6'	TCZNSEED	DFHZC4924	2 3 5 7 24
X'B7'	TCZASINC	DFHZC4925	2 3 5 7 24
X'B8'	TCZEVBAD	DFHZC4926	2 3 5 7 24
X'B9'	TCZFMH12	DFHZC4927	2 3 5 7 24
X'BB'	TCZSEXUC	DFHZC2418	2 3 7 9 10 11 23 24
X'BC'	TCZINIIR	DFHZC3410	2 3 9 10 11
X'BD'	TCZDESGM	DFHZC4928	7 24
X'BE'	TCZBFAIL	DFHZC4944	2 3 5 24
X'BF'	TCZCPFAL	DFHZC3490	7 24
X'C0'	TCZDWEGF	DFHZC3499	None
X'C1'	TCZSRCAT	DFHZC2400	2 3 7 9 10 11 23 24
X'C2'	TCZLUINP	DFHZC3486	7 24
X'C3'	TCZCPFAL	DFHZC3490	24
X'C5'	TCZSRCNA	DFHZC2427	2
X'C6'	TCZPASSD	DFHZC3484	None
X'C7'	TCZPSPRE	DFHZC3485	7 24
X'C8'	TCZLUINH	DFHZC3489	7 18 24
X'C9'	TCZNPSAU	DFHZC3487	7 24
X'CB'	TCZSRCTC	DFHZC2431	2 3 9 10 11
X'CC'	TCZSRCCI	DFHZC2451	2 3 9 10 11
X'CD'	TCZSRCCX	DFHZC2454	2 3 9 10 11
X'CE'	TCZVHOLD	DFHZC3469	7 9 10 11 24
X'CF'	TCZVRNOP	DFHZC3471	7 9 10 11 24
X'D0'	TCZTXCS	DFHZC2409	2 3 7 9 10 11 15 24
X'D1'	TCZTXCU	DFHZC2410	2 3 7 9 10 11 24
X'D3'	TCZDMPD	DFHZC2463	None
X'D4'	TCZCXRR	DFHZC2453	1 2 3 9 10

Table 52. Messages issued and flags set by DFHZNAC for specific error codes (continued)

Error code	Symbolic label	Message	Action flags set
X'D5'	TCZCXE2	DFHZC2452	3 7 9 10 11 18 24
X'D6'	TCZSXC2	DFHZC2441	None
X'D7'	TCZSXC1	DFHZC2440	None
X'D8'	TCZRNCH	DFHZC2457	2 3 7 9 10 11 24
X'D9'	TCZYX43	DFHZC2469	2 3 9 10 11
X'DA'	TCZSXC3	DFHZC2470	7 9 10 11 24
X'DB'	TCZPIPL	DFHZC2117	7 9 10 11 23 24
X'DC'	TCZPXE1	DFHZC2442	None
X'DD'	TCZPXE2	DFHZC2458	None
X'DE'	TCZPIPP	DFHZC2119	7 9 10 11 23 24
X'DF'	TCZDMGF	DFHZC3482	None
X'E0'	TCZDMSN	DFHZC2411	None
X'E1'	TCZDMRA	DFHZC2412	None
X'E2'	TCZDMCL	DFHZC2413	2
X'E3'	TCZCNCL	DFHZC2485	3 9 10 11
X'E4'	TCZAIER	DFHZC2433	None
X'E6'	TCZDMLG	DFHZC2404	None
X'E8'	TCZDMSLE	DFHZC3416	2 3
X'E9'	TCZSTIND	DFHZC2102	3
X'EA'	TCZSTLER	DFHZC3432	2 3
X'EB'	TCZSTRMH	DFHZC3428	3
X'EC'	TCZSTRMM	DFHZC3429	2 3 7
X'ED'	TCZSTON	DFHZC3430	2 3
X'EF'	TCZSTIN	DFHZC3431	2 3
X'F1'	TCZBDMOD	DFHZC4931	7 18 24
X'F2'	TCZEXRVT	DFHZC2469	2 3 9 10 11
X'F3'	TCZICTYP	DFHZC4932	2 3 7 24
X'F4'	TCZIDBA	DFHZC4933	2 3 24
X'F5'	TCZISYNL	DFHZC4934	2 3 7 24
X'F6'	TCZIUOW	DFHZC4935	2 3 7 24
X'F7'	TCZIFMHL	DFHZC4936	2 3 7 24
X'F8'	TCZFSMRB	DFHZC4943	3 7 9 10 11 24
X'F9'	TCZINVAT	DFHZC4946	2 3 7 24
X'FA'	TCZLUSEC	DFHZC4947	2 3 7 24
X'FB'	TCZPSUNB	DFHZC0125	7
X'FC'	TCZPSOPN	DFHZC0131	7
X'FD'	TCZPSRC	DFHZC0146	7
X'FE'	TCZPSRF	DFHZC0150	3 6 7 9 10 11 15 24
X'FF'	TCZPSPE	DFHZC0149	7

Notes:

1. See message DFHZC2497 or DFHZC3493, depending on the device type.
2. "Good morning" message to be sent.
3. Cancel task, and close z/OS Communications Server session owing to quick close or abend.

CICS messages associated with z/OS Communications Server errors

Table 53. CICS messages associated with z/OS Communications Server errors

Message	Symbolic label	Error code	Action flags set
DFHZC0101	TCZPRDTO	X'5D'	None
DFHZC0125	TCZPSUNB	X'FB'	7
DFHZC0131	TCZPSOPN	X'FC'	7
DFHZC0144	TCZPSRAC	X'7A'	7 11
DFHZC0145	TCZPSRAF	X'79'	3 6 7 9 10 11 24
DFHZC0146	TCZPSRC	X'FD'	7
DFHZC0147	TCZPSLE	X'71'	3 6 7 9 10 11 24
DFHZC0148	TCZPSVF	X'72'	7 9 10 11 24
DFHZC0149	TCZPSPE	X'FF'	7
DFHZC0150	TCZPSRF	X'FE'	3 6 7 9 10 11 15 24
DFHZC0155	TCZPSAF	X'6C'	3 6 7 9 10 11 24
DFHZC0156	TCZPSAR	X'6D'	7
DFHZC0157	TCZPSANR	X'7C'	3 7 9 10 11 24
DFHZC2102	TCZSTIND	X'E9'	3
DFHZC2117	TCZPIPL	X'DB'	7 9 10 11 23 24
DFHZC2119	TCZPIPP	X'DE'	7 9 10 11 23 24
DFHZC2400	TCZSRCAT	X'C1'	2 3 7 9 10 11 23 24
DFHZC2401	TCZRPLAC	X'B1'	2 3 7 9 10 11 23 24
DFHZC2402	TCZRSNE	X'B4'	3 11
DFHZC2403	TCZSRCBF	X'11'	2 5 7 18 24
DFHZC2404	TCZLRCER	X'14'	2 3 7 9 10 11 23 24
DFHZC2404	TCZLRCNR	X'18'	2 3 7 9 10 11 23 24
DFHZC2404	TCZDMLG	X'E6'	None
DFHZC2405	TCZSRCTU	X'10'	18
DFHZC2406	TCZSRCTS	X'19'	9 10 11 18
DFHZC2407	TCZSRCPF	X'15'	2 3 7 9 10 11 24
DFHZC2408	TCZSRCVE	X'1A'	2 3 7 9 10 11 24
DFHZC2408	TCZSRCV2	X'1E'	2 3 7 9 10 11 24
DFHZC2409	TCZTXCS	X'D0'	2 3 7 9 10 11 15 24
DFHZC2410	TCZTXCU	X'D1'	2 3 7 9 10 11 24
DFHZC2411	TCZDMSN	X'E0'	None
DFHZC2412	TCZDMRA	X'E1'	None
DFHZC2413	TCZDMCL	X'E2'	2

Table 53. CICS messages associated with z/OS Communications Server errors (continued)

Message	Symbolic label	Error code	Action flags set
DFHZC2414	TCZSRCSP	X'80'	None
DFHZC2416	TCZSRCVH	X'13'	7 18 24
DFHZC2417	TCZSRCVI	X'1D'	2 7 24
DFHZC2417	TCZVTAMI	X'20'	None
DFHZC2418	TCZSEXUC	X'BB'	2 3 7 9 10 11 23 24
DFHZC2419	TCZSSXUC	X'82'	2 3 7 9 10 11 23 24
DFHZC2420	TCZSAXUC	X'B5'	2 3 7 9 10 11 23 24
DFHZC2421	TCZUNCMD	X'60'	2 3 7 9 10 11 24
DFHZC2422	TCZLGCER	X'90'	1 2 3 6 9 10 11 23 24
DFHZC2423	TCZSDSE5	X'74'	3 7 9 10 11 24
DFHZC2424	TCZSESE1	X'75'	3 7 9 10 11 15 24
DFHZC2425	TCZSDAUC	X'B2'	3 7 9 10 11 15 24
DFHZC2426	TCZRACES	X'94'	2 3 9 10 11 22
DFHZC2427	TCZSRCNA	X'C5'	2
DFHZC2428	TCZSDSE6	X'92'	3 9 11
DFHZC2429	TCZRSTLE	X'91'	3 10 11
DFHZC2430	TCZSDRE2	X'78'	3 9 11 22
DFHZC2431	TCZSRCTC	X'CB'	2 3 9 10 11
DFHZC2432	TCZSSXNR	X'81'	None
DFHZC2433	TCZAIER	X'E4'	None
DFHZC2434	TCZCPYNS	X'43'	3 11
DFHZC2435	TCZRVSZ1	X'96'	3 7 10 11 24
DFHZC2436	TCZRVSZ3	X'97'	3 10 11
DFHZC2437	TCZSDSE4	X'73'	3 9 11
DFHZC2438	TCZRVSZ2	X'A1'	3 10 11
DFHZC2439	TCZACT01	X'98'	2 18
DFHZC2440	TCZSXC1	X'D7'	None
DFHZC2441	TCZSXC2	X'D6'	None
DFHZC2442	TCZPXE1	X'DC'	None
DFHZC2443	TCZROCT	X'59'	2 3 7 9 10 11 24
DFHZC2444	TCZBKTSE	X'A3'	2 3 7 9 10 11 24
DFHZC2445	TCZSDSE8	X'95'	3 9 11
DFHZC2446	TCZSSXIB	X'84'	2 3 7 9 10 11 23 24
DFHZC2447	TCZDOMCF	X'9A'	3 9 10 11 23
DFHZC2448	TCZINVRR	X'65'	2 3 7 10 11 22 23 24
DFHZC2449	TCZBOEB	X'A7'	2 3 7 11 18 22 24
DFHZC2450	TCZSSXAR	X'83'	None
DFHZC2451	TCZSRCCI	X'CC'	2 3 9 10 11
DFHZC2452	TCZCXE2	X'D5'	3 7 9 10 11 18 24
DFHZC2453	TCZCXRR	X'D4'	1 2 3 9 10

Table 53. CICS messages associated with z/OS Communications Server errors (continued)

Message	Symbolic label	Error code	Action flags set
DFHZC2454	TCZSRCCX	X'CD'	2 3 9 10 11
DFHZC2455	TCZRACET	X'93'	2 3 9 10 11
DFHZC2456	TCZSRCDE	X'44'	2 3 7 9 10 11 24
DFHZC2457	TCZRNCH	X'D8'	2 3 7 9 10 11 24
DFHZC2458	TCZPXE2	X'DD'	None
DFHZC2459	TCZSDSE7	X'99'	3 9 11
DFHZC2463	TCZDMPD	X'D3'	None
DFHZC2467	TCZLEXCI	X'88'	2 3 7 9 10 11 23 24
DFHZC2468	TCZLEXUS	X'89'	2 3 7 9 10 11 24
DFHZC2469	TCZYX43	X'D9'	2 3 9 10 11
DFHZC2469	TCZEXRVT	X'F2'	2 3 9 10 11
DFHZC2470	TCZSXC3	X'DA'	7 9 10 11 24
DFHZC2471	TCZFMHLE	X'A8'	2 3 4 7 10 11 22 24
DFHZC2472	TCZRACRF	X'A9'	11
DFHZC2473	TCZSDSE9	X'AA'	3 9 11
DFHZC2485	TCZCNCL	X'E3'	3 9 10 11
DFHZC2486	TCZRACNL	X'9B'	3
DFHZC2487	TCZLGNA	X'76'	3
DFHZC2488	TCZDMRY	X'77'	None
DFHZC2489	TCZINCPY	X'40'	3 9 11
DFHZC2490	TCZTOLRQ	X'41'	2 3 7 9 10 11 15 24
DFHZC2497	TCZUNPRT	X'42'	None
DFHZC2498	TCZICPUT	X'4B'	None
DFHZC3400	TCZCHMX	X'45'	3 10 11 22
DFHZC3402	TCZOCIR	X'46'	3 9 10 11
DFHZC3409	TCZUNEGR	X'85'	2 3 7 9 10 11 23 24
DFHZC3410	TCZINIIR	X'BC'	2 3 9 10 11
DFHZC3416	TCZDMSLE	X'E8'	2 3
DFHZC3417	TCZSDRE3	X'50'	3 7 9 10 11 24
DFHZC3418	TCZBDPRI	X'51'	3 7 9 10 11 24
DFHZC3419	TCZBDUAC	X'52'	2 3 5 7
DFHZC3420	TCZBDTOS	X'53'	7 20
DFHZC3421	TCZSBIRV	X'5A'	7 20
DFHZC3422	TCZNSP01	X'5B'	2 3 7 9 10 11 18 24
DFHZC3424	TCZNSP02	X'5C'	7 9 10 11 15 24
DFHZC3428	TCZSTRMH	X'EB'	3
DFHZC3429	TCZSTRMM	X'EC'	2 3 7
DFHZC3430	TCZSTON	X'ED'	2 3
DFHZC3431	TCZSTIN	X'EF'	2 3
DFHZC3432	TCZSTLER	X'EA'	2 3

Table 53. CICS messages associated with z/OS Communications Server errors (continued)

Message	Symbolic label	Error code	Action flags set
DFHZC3433	TCZERMGR	X'58'	7
DFHZC3434	TCZUNBIS	X'54'	2 3 7 9 10 11 24
DFHZC3440	TCZEMWBK	X'55'	None
DFHZC3441	TCZVTAMO	X'63'	None
DFHZC3442	TCZVTAMK	X'67'	None
DFHZC3443	TCZVTAMA	X'64'	None
DFHZC3444	TCZTIOAE	X'6A'	1 2 3 7 9 10 11 24
DFHZC3452	TCZSIGR	X'66'	None
DFHZC3454	TCZBRUAC	X'5E'	2 3 5 7 18 24
DFHZC3455	TCZBDSQP	X'5F'	2 3 5 7 18 24
DFHZC3461	TCZOPSIN	X'48'	7 , 8
DFHZC3462	TCZCLSIN	X'49'	7
DFHZC3463	TCZOPACB	X'4A'	None
DFHZC3464	TCZRELIS	X'57'	7 20
DFHZC3465	TCZRSPER	X'9D'	1 2 3 7 9 10 11 23
DFHZC3466	TCZSEXOS	X'69'	7 20 23
DFHZC3468	TCZCLRRV	X'70'	7 9 10 11 15 24
DFHZC3469	TCZVHOLD	X'CE'	7 9 10 11 24
DFHZC3470	TCZXSHRA	X'37'	9 10 11 24
DFHZC3470	TCZLUERR	X'AB'	7 9 10 11 24
DFHZC3471	TCZVRNOP	X'CF'	7 9 10 11 24
DFHZC3472	TCZDEVND	X'9E'	None
DFHZC3473	TCZSLSRL	X'4D'	None
DFHZC3474	TCZVRDAC	X'AC'	7 9 10 11 24
DFHZC3475	TCZNRLUF	X'AD'	7 9 10 11 24
DFHZC3476	TCZRCLUF	X'AE'	7 9 10 11 24
DFHZC3477	TCZCLEAN	X'AF'	7 9 10 11 24
DFHZC3479	TCZUNBFE	X'4E'	2 3 7 9 10 11 24
DFHZC3480	TCZNOISC	X'A0'	7 23 24
DFHZC3481	TCZDSPCL	X'4C'	2 3 7 9 10 11 24
DFHZC3482	TCZDMGF	X'DF'	None
DFHZC3484	TCZPASSD	X'C6'	None
DFHZC3485	TCZPSPRE	X'C7'	7 24
DFHZC3486	TCZLUINP	X'C2'	7 24
DFHZC3487	TCZNPSAU	X'C9'	7 24
DFHZC3488	TCZXUVAR	X'3C'	2 3 7 9 10 11 24
DFHZC3489	TCZLUINH	X'C8'	7 18 24
DFHZC3490	TCZCPFAL	X'C3'	7 24
DFHZC3491	TCZEXRO	X'B0'	7 15 24
DFHZC3492	TCZDMIT	X'16'	None

Table 53. CICS messages associated with z/OS Communications Server errors (continued)

Message	Symbolic label	Error code	Action flags set
DFHZC3495	TCZNOTNA	X'6B'	7 24
DFHZC3499	TCZDWEGF	X'C0'	None
DFHZC4902	TCZLUCF1	X'21'	3 7 9 10 11 24
DFHZC4903	TCZLUCF2	X'22'	3 7 9 10 11 24
DFHZC4904	TCZFSMBE	X'23'	3 7 10 11 9 24
DFHZC4905	TCZFSMCS	X'24'	3 7 10 11 9 24
DFHZC4906	TCZFSMCR	X'25'	3 7 10 11 9 24
DFHZC4907	TCZSDLER	X'26'	3 7 10 11 9 24
DFHZC4909	TCZRVLER	X'28'	3 7 10 11 9 24
DFHZC4910	TCZRVLRB	X'29'	3 7 10 11 9 24
DFHZC4911	TCZRLPEX	X'2A'	2 3 7 9 10 11 24
DFHZC4912	TCZRLPBD	X'2B'	2 3 7 9 10 11 24
DFHZC4913	TCZRLPDR	X'2C'	2 3 7 9 10 11 24
DFHZC4914	TCZRLPIL	X'2D'	2 3 7 9 10 11 24
DFHZC4915	TCZRLPEC	X'2E'	2 3 7 9 10 11 24
DFHZC4916	TCZRLPRR	X'2F'	2 3 7 9 10 11 24
DFHZC4917	TCZRLPIF	X'30'	2 3 7 9 10 11 24
DFHZC4918	TCZRLPIR	X'31'	2 3 7 9 10 11 24
DFHZC4919	TCZIVIND	X'33'	2 3 7 9 10 11 24
DFHZC4920	TCZIVDAT	X'34'	2 3 7 9 10 11 24
DFHZC4922	TCZRLXCL	X'32'	7 20
DFHZC4924	TCZNSEED	X'B6'	2 3 5 7 24
DFHZC4925	TCZASINC	X'B7'	2 3 5 7 24
DFHZC4926	TCZEVBAD	X'B8'	2 3 5 7 24
DFHZC4927	TCZFMH12	X'B9'	2 3 5 7 24
DFHZC4928	TCZDESGM	X'BD'	7 24
DFHZC4929	TCZBDBND	X'B3'	2 3 5 7 24
DFHZC4930	TCZRTMT	X'35'	2 3 7 9 10 11 24
DFHZC4931	TCZBDMOD	X'F1'	7 18 24
DFHZC4932	TCZICTYP	X'F3'	2 3 7 24
DFHZC4933	TCZIDBA	X'F4'	2 3 24
DFHZC4934	TCZISYNL	X'F5'	2 3 7 24
DFHZC4935	TCZIUOW	X'F6'	2 3 7 24
DFHZC4936	TCZIFMHL	X'F7'	2 3 7 24
DFHZC4937	TCZLUSRR	X'8A'	2 3 5 7 24
DFHZC4938	TCZLUSRF	X'8B'	2 3 5 7 24
DFHZC4939	TCZLUPUN	X'8C'	2 3 5 7 24
DFHZC4940	TCZLUSKN	X'8F'	2 3 5 7 24
DFHZC4941	TCZLUPLK	X'8D'	2 3 5 7 24
DFHZC4942	TCZLUPEX	X'8E'	2 3 5 7 24

Table 53. CICS messages associated with z/OS Communications Server errors (continued)

Message	Symbolic label	Error code	Action flags set
DFHZC4943	TCZFSMRB	X'F8'	3 7 9 10 11 24
DFHZC4944	TCZBFAIL	X'BE'	2 3 5 7 24
DFHZC4945	TCZPRGE	X'A2'	3 7 9 10 11 24
DFHZC4946	TCZINVAT	X'F9'	2 3 7 24
DFHZC4947	TCZLUSEC	X'FA'	2 3 7 24
DFHZC4949	TCZRABUS	X'7D'	2 3 7 9 10 11 24
DFHZC6590	TCZXRST	X'3F'	None
DFHZC6591	TCZXERR	X'3E'	7 9 10 11 15 24
DFHZC6593	TCZXSBC	X'3B'	None
DFHZC6594	TCZXSHR	X'3A'	7 24
DFHZC6595	TCZXSABN	X'39'	2 3 5 7 24
DFHZC6596	TCZXSWAS	X'38'	2 3 7 15 24
DFHZC6598	TCZXRFVS	X'56'	None

DFHZNAC—default actions for system sense codes

Table 54 shows the message issued and action flags set by DFHZNAC for each inbound system sense code received. See the *CICS Messages and Codes Vol 1* manual for a description of the conditions that correspond to the system sense codes. The figures in the "**Action flags set**" column are translated into bit settings and explained in Table 55 on page 908.

Table 54. Messages issued and flags set by DFHZNAC for specific sense codes

Sense code	Message	Action flags set
X'0001' [1]	DFHZC3401	2
X'0002' [1]	DFHZC3415	2, 3, 10, 11
X'0003' [1]	DFHZC3449	None
X'0004' [1]	DFHZC3450	None
X'0007' [1]	DFHZC3451	None [2]
X'00FF'	DFHZC3446	2, 3, 7, 9, 10, 11, 23, 24
X'0801'	DFHZC2476	3, 9, 10, 11
X'0802'	DFHZC2461	None
X'0806'	DFHZC3426	None
X'0807'	DFHZC3411	None
X'080B'	DFHZC2462	2, 3, 7, 9, 10, 11, 15, 24
X'080E'	DFHZC3448	23
X'080F'	DFHZC3436	9, 10, 11
X'0811'	DFHZC2464	9, 10, 11
X'0812'	DFHZC2465	2, 3
X'081B'	DFHZC2483	2, 3 [3]
X'081C'	DFHZC2466	2, 3, 9, 10, 11
X'0824'	DFHZC2475	3, 9, 10, 11

Table 54. Messages issued and flags set by DFHZNAC for specific sense codes (continued)

Sense code	Message	Action flags set
X'0825'	DFHZC2484	2, 3, 9, 10, 11
X'0826'	DFHZC3423	2, 3, 9, 10, 11
X'0827'	DFHZC2480	3
X'0829'	DFHZC3407	1, 2, 3, 7,10, 11, 24
X'082A'	None [4]	9
X'082B'	DFHZC3408	2, 3, 10, 11, 13
X'082D'	DFHZC3413	None
X'082E'	DFHZC3412	None
X'082F'	DFHZC3414	2, 3, 9, 10, 11
X'0831'	DFHZC3438	None
X'0833'	DFHZC3427	None
X'0847'	DFHZC3439	None
X'084A'	None [5]	None
X'084C'	DFHZC3467	9, 10, 11
X'0860'	DFHZC3459	None
X'0863'	DFHZC3460	9, 10, 11
X'0864'	DFHZC2475	3, 9, 10, 11
X'0865'	DFHZC2465	3, 9, 10, 11
X'0866'	DFHZC2475	3, 9, 10, 11
X'0867'	None [6]	9, 10, 11
X'0868'	DFHZC3456	2, 9, 10, 11
X'0869'	DFHZC3457	2, 9, 10, 11
X'08FF'	DFHZC3447	2, 3, 7, 9, 10, 11, 24
X'1000'	DFHZC3494	2, 3, 9, 10, 11
X'1001'	DFHZC2481	2, 3, 9, 10, 11, 14
X'1002'	DFHZC2481	2, 3, 9, 10, 11, 14
X'1003'	DFHZC2479	2, 3, 9, 10, 11, 14
X'1005'	DFHZC3406	2, 3, 4, 9, 10, 11, 14
X'1008'	DFHZC2478	None
X'1009'	DFHZC3458	2, 9, 10, 11
X'10FF'	DFHZC3446	2, 3, 7, 9, 10, 11, 23, 24
X'2003'	DFHZC3405	2, 3, 7, 9, 10, 11, 15, 24
X'20FF'	DFHZC3445	2, 3, 7, 9, 10, 11, 23, 24
X'400B'	DFHZC2477	1, 3, 11
X'40FF'	DFHZC3453	2, 3, 7, 9, 10, 11, 23, 24
X'8000'	DFHZC3435	2, 3, 7, 9, 10, 11, 18, 24
X'8005'	DFHZC3435	2, 3, 7, 9, 10, 11, 18, 24
X'80FF'	DFHZC3435	2, 3, 7, 9, 10, 11, 18, 23, 24
X'FFFF'	DFHZC2460	2, 3, 7, 9, 10, 11, 23, 24

Note:

1. The system sense code is in the form of an LUSTATUS command code.
2. No action flags are set if a task is attached or if outstanding operations are to complete. Otherwise, flag 21 is set.
3. Action flags 2 and 3 are set for negative response received for a SEND that requested a definite response.
4. Presentation space error.
5. Presentation error on read. Display buffer alteration, due to operator intervention, detected on a READ command to a compatibility-mode logical unit.
6. Function abend received from a device. A negative response to a chain was sent, but purged.

Action flag settings and meanings

Table 55 shows the "action flags" that can be set by DFHZNAC in the communication area passed to DFHZNEP. The flags set by DFHZNAC represent the default actions that will be taken if the settings are not changed by DFHZNEP.

The figures in the "**Flag**" column refer to those in columns 3 of Table 52 on page 895 and Table 54 on page 906.

Table 55. Meanings of action flags set by DFHZNAC

Flag	Field	Bit mask	Hex bit setting	Action
1	TWAOPT1	1...	X'80'	Print action flags
2		.1..	X'40'	Print z/OS Communications Server RPL
3		..1.	X'20'	Print TCTTE
4		...1	X'10'	Print TIOA
5		.... 1...	X'08'	Print BIND area
6		.... .1..	X'04'	System dump if no task attached
7		.... ..1.	X'02'	Print network-qualified name (NQNAME)
8		1	X'01'	Print TN3270 IP address (TNADDR)
9	TWAOPT2	1...	X'80'	Cancel any send for this terminal
10		.1..	X'40'	Cancel any receive for this terminal
11		..1.	X'20'	Abend any task attached to TCTTE
12		...1	X'10'	Cancel any task attached to TCTTE
13		.... 1...	X'08'	Good Morning message to be sent
14		.... .1..	X'04'	Purge any BMS pages for this TCTTE
15		.... ..1.	X'02'	SIMLOGON required
17	TWAOPT3	1...	X'80'	Set INTLOG now allowed

Table 55. Meanings of action flags set by DFHZNAC (continued)

Flag	Field	Bit mask	Hex bit setting	Action
18		.1..	X'40'	Set no internal general logons
20		...1	X'10'	Normal CLSDST (no reset allowed)
21		.... 1...	X'08'	Normal CLSDST (reset allowed)
22		.... .1..	X'04'	Send negative response
23		.... ..1.	X'02'	AOS - keep node out of service
24		1	X'01'	CLSDST node

Appendix D. Analysis of CICS restart information

When CICS shuts down, you can programmatically determine the type of shutdown that occurred, and whether it is safe to override the next restart type and perform a cold restart of the CICS system.

DFHRMUTL provides the SET_AUTO_START keywords to automatically restart a CICS system with a specific override. The keywords can include AUTOCOLD, which makes CICS perform a cold start. A CICS system that was shut down using a warm shutdown, and which had no indoubt, commit-failed, or backout-failed units of work (UOWs) keypoints at that time, can be restarted cold without loss of data integrity. However, if the system was not shut down in a controlled manner, or there are such units of work recorded on the system log, do not perform a cold restart, because system data integrity can be compromised.

To help you determine whether a cold start of the region system is appropriate or not, CICS provides the following information:

- The following DFHRMUTL summary information:
  - The next CICS start type found on the global catalog, DFHGCD.
  - If the next CICS start type is warm, counts of the units of work that were in an indoubt, commit-failed, or backout-failed state when CICS was shut down.
- A CICS global catalog record with the following VSAM KSDS key:

  ```
  X'00000011C4C6C8D9D4C4D440C4C6C8D9D4C4D46DD9C5E2E3C1D9E340'
  ```

  This record contains three fullword counts following the key:
  - The number of indoubt UOWs
  - The number of commit-failed UOWs
  - The number of backout-failed UOWs

  If any of these counts contains a number greater than zero, recovery information for the units of work is present on the CICS system log. In this case, a modification of the CICS autostart override record to AUTOCOLD is not appropriate, because system data integrity would be jeopardized.

  CICS deletes this global catalog record during restart, and writes it during a controlled shutdown. Therefore presence or absence of the record on the catalog indicates whether the CICS system was previously shut down warmly; that is, by the use of **CEMT PERFORM SHUTDOWN** or the **Shutdown** option from the CICS Explorer **Regions** operations view. If the record exists on the global catalog, and the three count fields are zero, it is safe to use DFHRMUTL to modify the autostart override record to specify AUTOCOLD.

  Here is an example of this record, showing the presence of one indoubt unit of work at the time CICS was shut down:

  ```
  KEY OF RECORD -
  00000011C4C6C8D9D4C4D440C4C6C8D9D4C4D46DD9C5E2E3C1D9E340

  00000011C4C6C8D9D4C4D440C4C6C8D9D4C4D46DD9C5E2E3C1D9E34000000001
  0000000000000000
  ```

  You can use the DFHRMRED Assembler DSECT in a batch program to map the global catalog record. DFHRMRED is found in the CICSTS52.CICS.SDFHMAC library.

For more information about the DFHRMUTL utility, see , in the *CICS Operations and Utilities Guide*.

Appendix E. Using the transient data write-to-terminal program (DFH$TDWT)

DFH$TDWT is a sample program that sends transient data messages to a terminal or printer. You can use it to send messages from a single transient data queue, or from several queues, to one terminal.

In the definition for the transient data queue, you can specify that particular categories of message (for example, those from the abnormal condition program and signon and sign-off messages) should be sent to destinations defined as indirect. If these indirect destinations are defined so that they refer to the same intrapartition queue with a transaction identifier and a trigger level of 1, the receipt of a single message in any of the specified categories causes the transaction to be started. The program thus invoked displays or prints the message. The transaction that invokes the DFH$TDWT sample program is TDWT.

To use the sample program, your CICS system must include automatic transaction initiation and an intrapartition transient data set. The source code for the DFH$TDWT sample program is provided in CICSTS52.CICS.SDFHSAMP, and the object code is provided in CICSTS52.CICS.SDFHLOAD.

For detailed information about defining transient data queues, see the *CICS Resource Definition Guide*.

DFH$TDWT—resource definitions required

To use the DFH$TDWT sample program as supplied, you need the following resource definitions installed on your CICS region:

- A program definition for the DFH$TDWT program
- A transaction definition for the TDWT transaction
- A terminal definition for the L86P terminal
- A definition for the intrapartition queue, L86P
- Definitions for the indirect intrapartition queues pointing to the L86P queue.

These required resource definitions are provided in the CICS-supplied group, DFH$UTIL. Add DFH$UTIL to your CICS startup group list.

However, you must define the other resources. Add a terminal definition for the L86P terminal to the CSD, and install it in your CICS region.

Appendix F. Uppercase translation

You can customize the translation of lower- and mixed-case characters and national characters that are input at a terminal. You can also translate operator messages produced by temporary storage data sharing.

Translating national characters to uppercase

In CICS, translation of terminal user-input to uppercase characters can be done either by using the UCTRAN option on the PROFILE and TYPETERM definitions, or by using the EXEC CICS SET TERMINAL(termid) UCTRANST command.

However, some languages have characters which are not part of the set of EBCDIC characters translated by UCTRAN, and so are never translated to uppercase, regardless of what you have specified on your resource definitions. To translate these national characters, you have two options:

- Use the XZCIN exit
- Create a new terminal control table (TCT), based on your current TCT (or on the dummy TCT, DFHTCTDY, if you have TCT=NO specified in the SIT), and modify the translation table in it.

Whichever method you use, the *Character Data Representation Architecture Level 1 - Registry* manual, SC09-1391-00, is a useful reference for information on code pages.

Using the XZCIN exit

XZCIN is described in "Exit XZCIN" on page 316. To use it for uppercase translation, you must supply your own translation routine, which is then invoked when terminal input occurs.

Using DFHTCTxx

To translate national characters which are not handled by UCTRAN, you can modify the translation table in the terminal control table.

About this task

If you use RDO for all your terminals and have TCT=NO specified in your SIT or its overrides, CICS uses the dummy TCT, DFHTCTDY, to create control blocks for RDO-defined and autoinstalled terminals. It is not recommended that you modify DFHTCTDY directly. Instead, take a copy of the DFHTCTDY source file, save it under a new name, and modify the copy. The steps you need to follow are:

Note: If you are already using a customized TCT rather than DFHTCTDY (that is, something other than 'NO' or 'DY' is specified on the SIT TCT parameter), you must add your translation code to the TCT you are using.

Procedure

1. Take a copy of the DFHTCTDY assembler source file. CICS provides this in the CICSTS52.CICS.SDFHSAMP library.
2. Modify the translation table in the source file, as shown in Figure 119 on page 916.

3. Save the source file as DFHTCTxx, where 'xx' is any 2–character suffix other than 'DY'.
4. Use the CICS-supplied DFHAUPLE job to assemble, define to SMP/E, and linkedit the new table.
5. Specify the 2–character suffix of your new TCT on the SIT TCT parameter.
6. Restart CICS, so that the new TCT takes effect.

Example

Figure 119 shows a suggested way to code the assembler source statements to translate your national characters.

```
         MACRO
         NATLANG
DFHUCTRT CSECT                          Resume UCTRAN table CSECT
.*
.* This example translates lowercase 'a' ( EBCDIC X'81') to
.* uppercase 'A' (EBCDIC X'C1') for a US code page.
.*
         ORG  TCZUCTAB+X'81'            Reset the counter to the
                                        character to be translated.
         DC   X'C1'                     Declare the replacement
                                        character as a constant.
.*
.* Repeat the above two statements for each extra character you want
.* to be translated.
.*
         ORG  ,                         Reset the location counter
&SYSLOC  LOCTR                          Resume previous location counter
         MEND                           End of macro definition
         DFHTCT TYPE=INITIAL,SUFFIX=xx,                                *
               MIGRATE=COMPLETE,                                       *
               ACCMETH=(VTAM),                                         *
               DUMMY=DUMMY
         NATLANG                        Execute NATLANG
         DFHTCT TYPE=FINAL
         END DFHTCTBA
```

Figure 119. Suggested coding for national language character translation

Translating TS data sharing messages to uppercase

CICS temporary storage (TS) data sharing uses AXM services to write operator messages. These messages are in mixed-case English; table AXMMSTAB is used to remove unprintable characters. If necessary, you can modify AXMMSTAB to convert the messages to uppercase English.

The modules that use AXM message services are AXMSI, AXMSC, and DFHXQMN. AXMSI and AXMSC are both in a linklist library; DFHXQMN is in the CICS authorized library. To convert TS data sharing messages to uppercase, modify the copy of AXMMSTAB used by each of these modules by using SPZAP with the following input (for each module):

```
NAME modulename AXMMSTAB
VER 0081 818283848586878889
VER 0091 919293949596979899
VER 00A2 A2A3A4A5A6A7A8A9
REP 0081 C1C2C3C4C5C6C7C8C9
REP 0091 D1D2D3D4D5D6D7D8D9
REP 00A2 E2E3E4E5E6E7E8E9
```

Appendix G. The example program for the XTSEREQ global user exit, DFH$XTSE

The example global user exit program, DFH$XTSE, shows you how to use CICS commands, XPI calls, and other functions in a global user exit program.

The example program shows you how to do the following tasks:

- Use EXEC CICS commands in a global user exit program
- Use EXEC CICS commands and XPI calls in the same exit program
- Modify the command parameter list in EXEC interface exits such as XTSEREQ, XICEREQ, and XFCREQ
- Modify Temporary Storage (TS) requests.

You must customize the DFH$XTSE program before using it in a production environment. The DFH$XTSE program is in the SDFHSAMP library.

Appendix H. Threadsafe XPI commands

Most, but not all, XPI commands are threadsafe. Issuing any of the non-threadsafe commands causes CICS to use the QR TCB to ensure serialization.

The XPI commands that are threadsafe are indicated in the command syntax diagrams in with the statement: “This command is threadsafe”, and are listed here.

Threadsafe commands

- DFHAPIQX INQ_APPLICATION_DATA
- DFHBRIQX INQUIRE_CONTEXT
- DFHDDAPX BIND_LDAP
- DFHDDAPX END_BROWSE_RESULTS
- DFHDDAPX FLUSH_LDAP_CACHE
- DFHDDAPX FREE_SEARCH_RESULTS
- DFHDDAPX GET_ATTRIBUTE_VALUE
- DFHDDAPX GET_NEXT_ATTRIBUTE
- DFHDDAPX GET_NEXT_ENTRY
- DFHDDAPX SEARCH_LDAP
- DFHDDAPX START_BROWSE_RESULTS
- DFHDDAPX UNBIND_LDAP
- DFHDSATX CHANGE_PRIORITY
- DFHDSSRX ADD_SUSPEND
- DFHDSSRX DELETE_SUSPEND
- DFHDSSRX RESUME
- DFHDSSRX SUSPEND
- DFHDSSRX WAIT_MVS
- DFHDUDUX SYSTEM_DUMP
- DFHJCJCX WRITE_JOURNAL_DATA
- DFHKEDSX START_PURGE_PROTECTION
- DFHKEDSX STOP_PURGE_PROTECTION
- DFHLDLDX ACQUIRE_PROGRAM
- DFHLDLDX DEFINE_PROGRAM
- DFHLDLDX DELETE_PROGRAM
- DFHLDLDX IDENTIFY_PROGRAM
- DFHLDLDX RELEASE_PROGRAM
- DFHLGPAX INQUIRE_PARAMETERS
- DFHLGPAX SET_PARAMETERS
- DFHMNMNX INQUIRE_MONITORING_DATA
- DFHMNMNX MONITOR
- DFHNQEDX DEQUEUE
- DFHNQEDX ENQUEUE
- DFHPGAQX INQUIRE_AUTOINSTALL
- DFHPGAQX SET_AUTOINSTALL
- DFHPGISX END_BROWSE_PROGRAM
- DFHPGISX GET_NEXT_PROGRAM
- DFHPGISX INQUIRE_CURRENT_PROGRAM
- DFHPGISX INQUIRE_PROGRAM
- DFHPGISX SET_PROGRAM
- DFHPGISX START_BROWSE_PROGRAM
- DFHSAIQX INQUIRE_SYSTEM
- DFHSAIQX SET_SYSTEM
- DFHSMMCX GETMAIN

- DFHSMMCX FREEMAIN
- DFHSMMCX INQUIRE_ELEMENT_LENGTH
- DFHSMMCX INQUIRE_TASK_STORAGE
- DFHSMSRX INQUIRE_ACCESS
- DFHSMSRX INQUIRE_SHORT_ON_STORAGE
- DFHSMSRX SWITCH_SUBSPACE
- DFHTRPTX TRACE_PUT
- DFHXMCLX INQUIRE_TCLASS
- DFHXMIQX INQUIRE_TRANSACTION
- DFHXMIQX SET_TRANSACTION
- DFHXMSRX INQUIRE_DTRTRAN
- DFHXMSRX INQUIRE_MXT
- DFHXMXDX INQUIRE_TRANDEF

Non-threadsafe commands

- DFHDUDUX TRANSACTION_DUMP

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:

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

The following paragraph does not apply in the United Kingdom or any other country where such provisions are inconsistent with local law:

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

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

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN.

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

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

Privacy Policy Considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

CICSPlex SM Web User Interface :

For the WUI main interface: Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name and other personally identifiable information for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the WUI Data Interface: Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other personally identifiable information for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the WUI Hello World page: Depending upon the configurations deployed, this Software Offering may use session cookies that collect no personally identifiable information. These cookies cannot be disabled.

For CICS Explorer: Depending upon the configurations deployed, this Software Offering may use session and persistent preferences that collect each user's user name and password, for purposes of session management, authentication, and single sign-on configuration. These preferences cannot be disabled, although storing a user's password on disk in encrypted form can only be enabled by the user's explicit action to check a check box during sign-on.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www-01.ibm.com/software/info/product-privacy/.

Trademarks

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

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

Bibliography

CICS books for CICS Transaction Server for z/OS

General

CICS Transaction Server for z/OS Program Directory, GI13-3326
CICS Transaction Server for z/OS What's New, GC34-7302
CICS Transaction Server for z/OS Upgrading from CICS TS Version 3.1, GC34-7296
CICS Transaction Server for z/OS Upgrading from CICS TS Version 3.2, GC34-7297
CICS Transaction Server for z/OS Upgrading from CICS TS Version 4.1, GC34-7298
CICS Transaction Server for z/OS Upgrading from CICS TS Version 4.2, GC34-7299
CICS Transaction Server for z/OS Upgrading from CICS TS Version 5.1, GC34-7300
CICS Transaction Server for z/OS Installation Guide, GC34-7279

Access to CICS

CICS Internet Guide, SC34-7281
CICS Web Services Guide, SC34-7301

Administration

CICS System Definition Guide, SC34-7293
CICS Customization Guide, SC34-7269
CICS Resource Definition Guide, SC34-7290
CICS Operations and Utilities Guide, SC34-7285
CICS RACF Security Guide, SC34-7288
CICS Supplied Transactions, SC34-7292

Programming

CICS Application Programming Guide, SC34-7266
CICS Application Programming Reference, SC34-7267
CICS System Programming Reference, SC34-7294
CICS Front End Programming Interface User's Guide, SC34-7277
CICS C++ OO Class Libraries, SC34-7270
CICS Distributed Transaction Programming Guide, SC34-7275
CICS Business Transaction Services, SC34-7268
Java Applications in CICS, SC34-7282

Diagnosis

CICS Problem Determination Guide, GC34-7287
CICS Performance Guide, SC34-7286
CICS Messages and Codes Vol 1, GC34-7283
CICS Messages and Codes Vol 2, GC34-7284
CICS Diagnosis Reference, GC34-7274
CICS Recovery and Restart Guide, SC34-7289
CICS Data Areas, GC34-7271
CICS Trace Entries, SC34-7295
CICS Debugging Tools Interfaces Reference, GC34-7273

Communication

CICS Intercommunication Guide, SC34-7280
CICS External Interfaces Guide, SC34-7276

Databases

CICS DB2 Guide, SC34-7272

CICS IMS Database Control Guide, SC34-7278

CICS Shared Data Tables Guide, SC34-7291

CICSPlex SM books for CICS Transaction Server for z/OS

General

CICSPlex SM Concepts and Planning, SC34-7306
CICSPlex SM Web User Interface Guide, SC34-7316

Administration and Management

CICSPlex SM Administration, SC34-7303
CICSPlex SM Operations Views Reference, SC34-7312
CICSPlex SM Monitor Views Reference, SC34-7311
CICSPlex SM Managing Workloads, SC34-7309
CICSPlex SM Managing Resource Usage, SC34-7308
CICSPlex SM Managing Business Applications, SC34-7307

Programming

CICSPlex SM Application Programming Guide, SC34-7304
CICSPlex SM Application Programming Reference, SC34-7305

Diagnosis

CICSPlex SM Resource Tables Reference Vol 1, SC34-7314
CICSPlex SM Resource Tables Reference Vol 2, SC34-7315
CICSPlex SM Messages and Codes, GC34-7310
CICSPlex SM Problem Determination, GC34-7313

Other CICS publications

The following publications contain further information about CICS, but are not provided as part of CICS Transaction Server for z/OS, Version 5 Release 2.

Designing and Programming CICS Applications, SR23-9692

CICS Application Migration Aid Guide, SC33-0768

CICS Family: API Structure, SC33-1007

CICS Family: Client/Server Programming, SC33-1435

CICS Family: Interproduct Communication, SC34-6853

CICS Family: Communicating from CICS on System/390, SC34-6854

CICS Transaction Gateway for z/OS Administration, SC34-5528

CICS Family: General Information, GC33-0155

CICS 4.1 Sample Applications Guide, SC33-1173

CICS/ESA 3.3 XRF Guide , SC33-0661

Other IBM publications

The following publications contain information about related IBM products.

MVS books

z/OS MVS Authorized Assembler Services Guide, SA22-7608
z/OS MVS Authorized Assembler Services Reference Volume 1, SA22-7609
z/OS MVS Authorized Assembler Services Reference Volume 2, SA22-7610

z/OS MVS Authorized Assembler Services Reference Volume 3, SA22-7611
z/OS MVS Authorized Assembler Services Reference Volume 4, SA22-7612
z/OS MVS Data Areas Volume 1, GA22-7581
z/OS MVS Data Areas Volume 2, GA22-7582
z/OS MVS Data Areas Volume 3, GA22-7583
z/OS MVS Data Areas Volume 4, GA22-7584
z/OS MVS Data Areas Volume 5, GA22-7585
z/OS Resource Measurement Facility User's Guide, SC33-7990
z/OS MVS System Management Facilities (SMF), SA22-7630

VTAM books

VTAM Network Implementation Guide, SC31-6494
VTAM Programming, SC31-6496

Other related books

z/Architecture Principles of Operation, SA22-7832
IMS Application Programming API Reference, SC18-9699
IMS Communications and Connections Guide, SC18-9703
OS/390 Security Server External Security Interface (RACROUTE) Macro Reference for MVS, GC28-1922
OS/390 Security Server (RACF) Security Administrator's Guide, SC28-1915
Service Level Reporter Version 3 General Information, GH19-6529
SNA Formats, GA27-3136
SNA Sessions Between Logical Units, GC20-1868

Accessibility

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

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

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

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

Index

Special characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

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

CICS Transaction Server for z/OS
Version 5 Release 2
Customization Guide

Publication No. SC34-7269-00

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

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

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

Comments:

Thank you for your support.

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

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

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC34-7269-00

Cut or Fold
Along Line

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

PLACE
POSTAGE
STAMP
HERE

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

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

Cut or Fold
Along Line

SC34-7269-00