CICS Transaction Server for z/OS
5.5

Developing CICS System Programs

Note

Before using this information and the product it supports, read the information in "Notices" on page 383.

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

Contents

About this PDF

This PDF 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. You might also need the two companion reference PDFs: *XPI Function Reference* and *Global User Exit Reference*. Before CICS TS V5.4, the information in this PDF and the two reference PDFs was in one PDF called the *Customization Guide*. Other PDFs, listed below, describe how to customize certain areas of CICS and you might need to refer to those as well as this PDF. (In IBM Knowledge Center, all this information is under one section called "Developing system programs".)

Customization information for areas of CICS is in the following PDFs:

- SOAP and JSON is in *Web Services Guide* .
- Front End Programming Interface is in the *Front End Programming Interface User's Guide*.
- Shared data tables are in the *Shared Data Tables Guide*.
- CICSPlex SM is in *CICSPlex SM Administration*.
- External security manager is in *RACF Security Guide*

For details of the terms and notation used in this book, see Conventions and terminology used in the CICS documentation in IBM Knowledge Center.

Date of this PDF

This PDF was created on 2024-04-22 (Year-Month-Date).

Chapter 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.

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.

CICS provides two sets of sample and example global user exit programs. For more information, see Samples.

Writing global user exit programs

You must write global user exit programs in assembler language and they must be quasi-reentrant. 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 quasi-reentrant 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 quasi-reentrant programs, see Multithreading: Reentrant, quasi-reentrant, and threadsafe programs .

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 z/Architecture Principles of Operation.
- The contents of the access registers are unpredictable. For information about access registers, see z/Architecture Principles of Operation.
- 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 7.

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 52 for information).

For an example of how to use EXEC CICS commands and XPI calls in the same global user exit program, see Global user exit sample program DFH$XTSE.

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 Transferring data between programs using channels.

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.

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 5. 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 Global user exit foundation samples.

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.

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 "The user exit programming interface (XPI)" on page 48. 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. 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 10.

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 1. 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

Table 1. TCB indicators in DFHUEPAR (continued)		
Symbolic value	**2-byte code**	**Description**
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 "The user exit programming interface (XPI)" on page 48. 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 "The user exit programming interface (XPI)" on page 48.

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 10
- 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 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 Global user exit foundation samples and Specific exit program samples.

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 **CICS operations** > **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 Exit-related commands. 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.

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.

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 1 on page 13 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 33. 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.

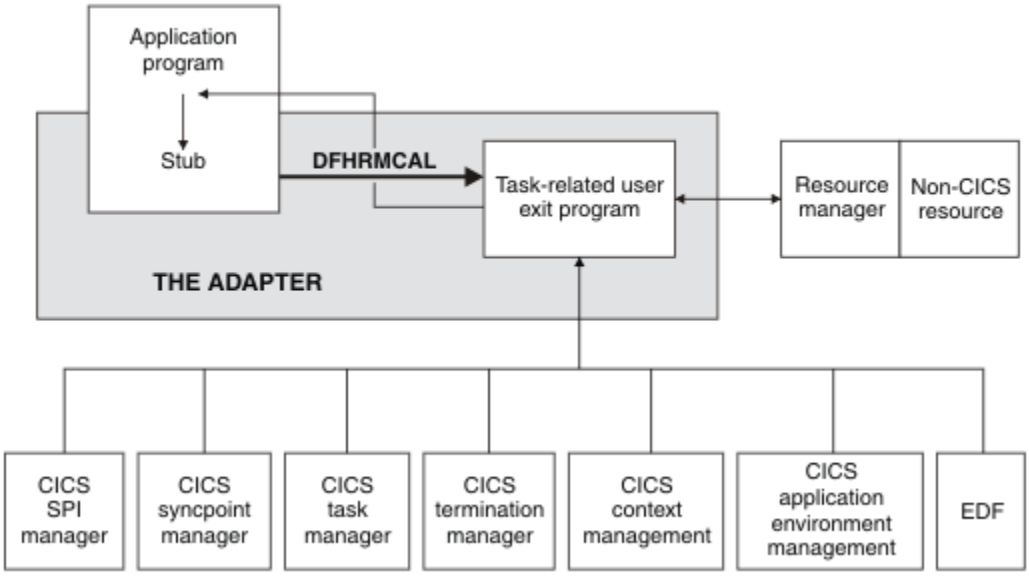


Figure 1. The adapter concept

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, DISABLE PROGRAM command, EXTRACT EXIT, and RESYNC ENTRYNAME.

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.

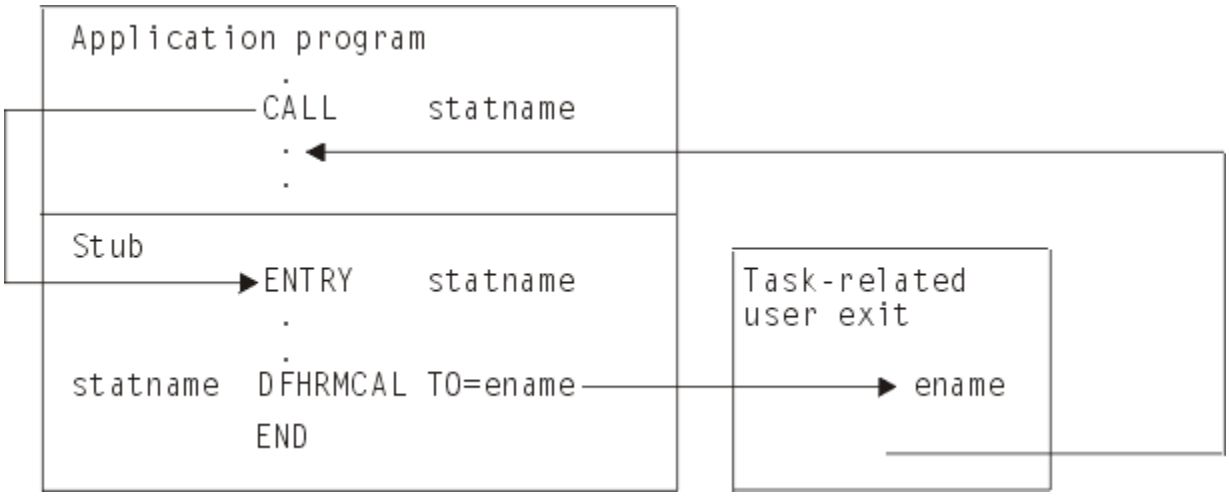


Figure 2. The stub concept

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 2 on page 14. 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 28 and “Using EDF with your task-related user exit program” on page 45.

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.

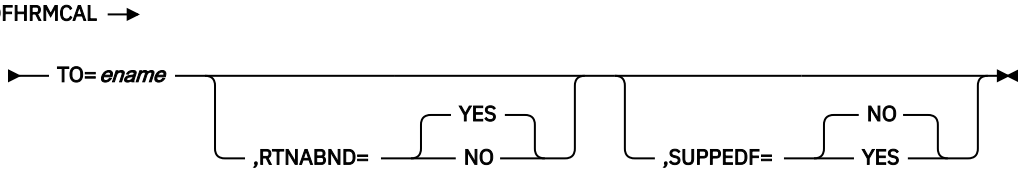
Note: If you specify SUPPEDF=YES on the DFHRMCAL macro, the "About to Execute" and "Command Execution Complete" screens relating to the invocation of the TRUE by DFHRMCAL are suppressed; in other words, DFHRMCAL becomes "invisible" to EDF. Specifying SUPPEDF=YES has no effect in determining whether EDF 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 24.

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).

TRUEs and types of TCB used

A task-related user exit (TRUE) can run on the main CICS QR TCB or an open TCB depending on the program attributes of the TRUE.

A Quasirent TRUE is enabled with the QUASIRENT option. Quasirent TRUEs are always invoked on the QR TCB instead of an open TCB. If a Quasirent TRUE is to invoke an external resource manager, it must manage its own set of 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.

A TRUE that is enabled with the THREADSAFE option will run on whatever TCB the task is running on at the time when the TRUE is invoked. This could be the QR TCB or an open TCB.

A TRUE that is enabled with both the THREADSAFE and OPENAPI options or with both the REQUIRED and OPENAPI options will always run on an L8 open TCB. A TRUE that is enabled with the REQUIRED but not the OPENAPI option will always run on a key 8 open TCB. This key 8 open TCB could be an L8, a T8, or an X8 open TCB, depending on the language of the calling application. For more information about open TCB modes, see Open TCB management.

Obligations of task-related user exits (TRUEs) running on open TCBs

If a TRUE runs on an open TCB, it is freed from the constraints imposed by the QR TCB and avoids the need to create and manage its own set of subtask TCBs. If an operating system wait issued by an external resource manager halts the open TCB, CICS continues processing on the QR TCB, and on other open TCBs. A TRUE running on an open TCB nevertheless has obligations both to the CICS system as a whole and to future users of the open TCB it is using.

Obligations of TRUEs running on L8 open TCBs

An L8 TCB is dedicated for use by the CICS task to which it is allocated to, 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 TRUE, and not that the TRUE has broken the threadsafe restrictions, which CICS would not detect.

An L8 TCB is not dedicated for use by a particular TRUE, but is used by all TRUEs that are invoked by the CICS task and that require an L8 TCB. The L8 TCB is also used by the threadsafe application code executed by the task.

Obligations of TRUEs running on T8 and X8 open TCBs

For XPLINK programs and for Java programs running in an OSGi JVM, X8 and T8 TCBs are respectively dedicated for to use by the CICS task at that link level. When control returns to a higher link level, the TCB is freed and is available for use by another CICS task.

For Java programs running in a Liberty JVM, the T8 TCB is dedicated to use by a CICS task for the lifetime of that task in the same way as an L8 TCB. If T8 TCBs remain clean, they can be used by subsequent tasks. And the application code that is executed by the calling task runs on the same TCB as the TRUE.

Threadsafe restrictions

A TRUE running on an open TCB must not treat the executing open TCB environment in such a way that it causes problems for:

- Other TRUEs running on open TCBs 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, A TRUE running on an open TCB 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, A TRUE running on an open TCB 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)
- A TRUE running on an open TCB 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.
- A TRUE running on an open TCB 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 a task-related user exit that runs on an open TCB

About this task

If a task-related user exit (TRUE) is enabled with options that indicate it always runs on an open TCB, CICS uses the following rules, based on the type of call, to determine the TCB on which it should invoke the TRUE.

Application program call (API)—UERTAPPL
For this call, if TRUEs are enabled to always run on an L8, they are called on an L8 TCB. If TRUEs are enabled to run on any key 8 TCB, they are invoked on an L8, T8, or X8.

CICS syncpoint manager call—UERTSYNC
For this call, if TRUEs are enabled to always run on an L8, they are called on an L8 TCB. If TRUEs are enabled to run on any key 8 TCB, they are invoked on an L8, T8, or X8.

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, tasks that run as Liberty threads invoke the TRUE on the T8 TCB that CICS provided to the Liberty ThreadPool. In all other environments, for performance reasons, an open TCB is not used and the TRUE is always invoked on the QR TCB.

UERTEOTR —End of task
For this call, if TRUEs are enabled to always run on an L8, they are called on an L8 TCB. If TRUEs are enabled to run on any key 8 TCB, they are invoked on an L8, T8, or X8.

EDF call—UERTFEDF
For this call, if TRUEs are enabled to always run on an L8, they are called on an L8 TCB. If TRUEs are enabled to run on any key 8 TCB, they are invoked on an L8, T8, or X8.

CICS SPI call—UERTSPI
For this call, for performance reasons, CICS always invokes the TRUE as a threadsafe TRUE and hence calls the TRUE on the TCB on which the task is currently running.

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, open TCBs are not used and CICS always calls the TRUE on the QR TCB.

Note: Even for call types that are invoked on an open TCB, it is possible that the open TCB could suffer an asynchronous abend and therefore not be available for subsequent use, with the following result:

- If CICS is unable to switch to the open TCB for an API call to a TRUE, CICS abends the transaction.
- If CICS is unable to switch to the open TCB for a syncpoint or end of task call, CICS invokes the TRUE on the QR TCB instead.

The TCB mode on which the TRUE is being called is provided in the second and third bytes of a three-byte-field address parameter identified by the **UEPTIND** symbolic name. 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.

In the Liberty environment, do not cache the user ID for later use. The user ID passed to the start of task TRUE may not reflect the eventual user ID that is established for the task. Subsequent TRUEs will provide the user ID.

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 28. All of the user exit parameter lists are summarized in "Summary of the task-related user exit parameter lists" on page 30.

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 23 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 36. 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 30. 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 36. 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 31.

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 38.

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 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
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.
- 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.

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 20 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 47).

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.

   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 42.

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 the task that is issuing the START command contains origin data adapter data.

UECON_ADAPTER_DATA_ON
(X'80') indicates that the task that is issuing the START command contains origin data adapter data, and therefore it is not the first (origin) adapter for this set of tasks.

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 3 on page 30.

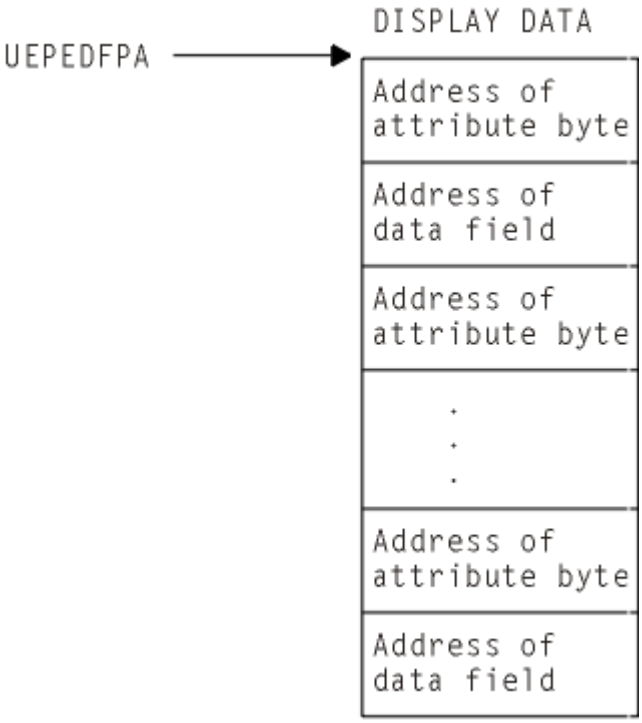


Figure 3. Display data parameter list

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.
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 4 on page 31 shows, in diagrammatic form, the relationships between the parameter lists that are discussed in the preceding sections.

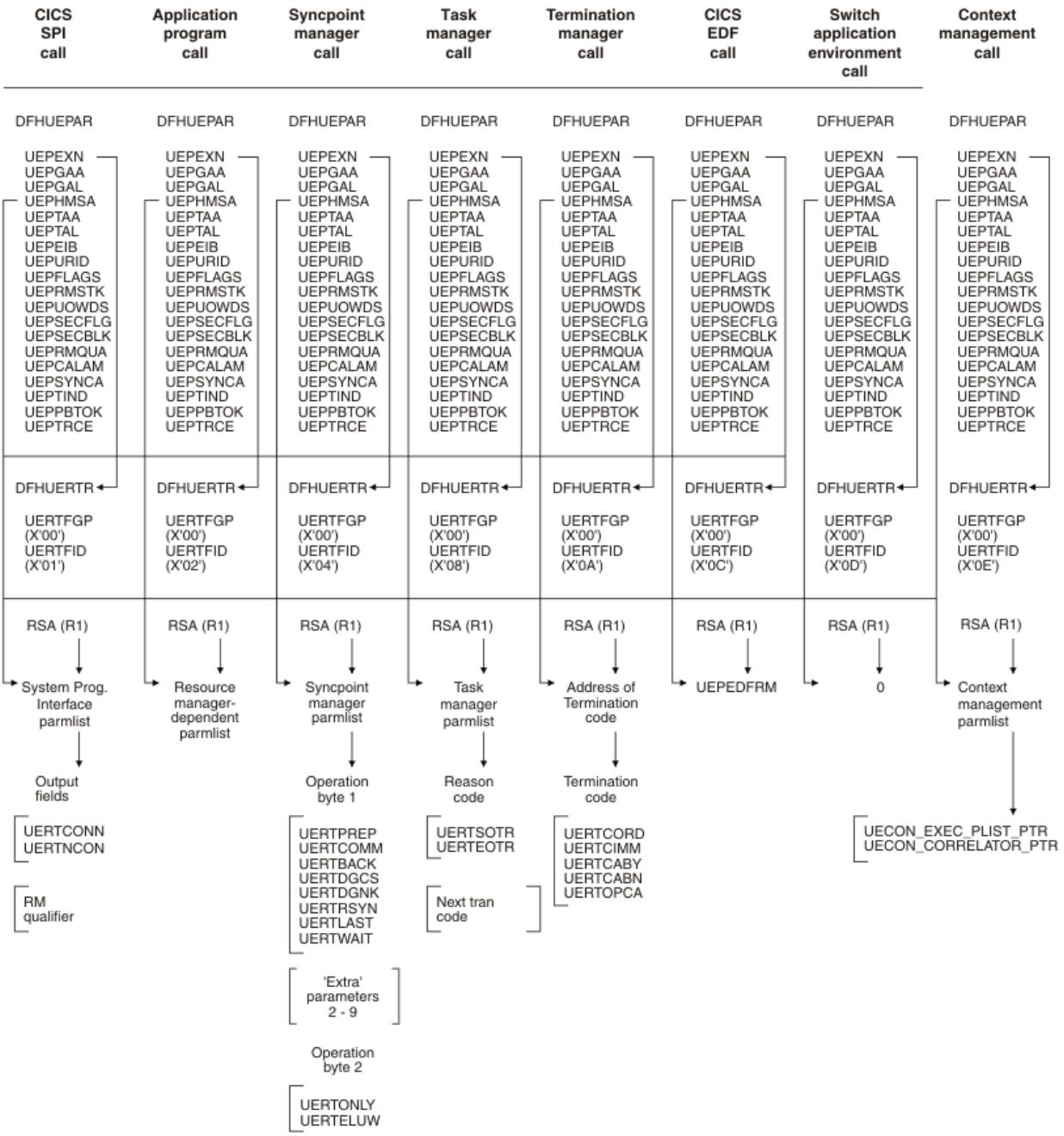


Figure 4. Task-related user exit parameter lists

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 3. Format of the schedule flag word

Byte	Setting	EXEC CICS ENABLE option	Comments
0	—	—	Reserved
1	—	—	Reserved
2 UEFDCON UEFDSWAE UEFDFEDF UEFDCTER UEFDTASK	 UEFMCON (X'40') UEFMSWAE (X'20') UEFMFEDF (X'10') UEFMCTER (X'04') UEFMTASK (X'01')	 — — FORMATEDF SHUTDOWN TASKSTART	 Bit mask for context management Bit mask for switch application environment Bit mask for EDF invocation Bit mask for termination manager Bit mask for task manager
3 UEFDSYNC UEFDAPPL UEFDSPI	 UEFMSYNC (X'10') UEFMAPPL (X'04') UEFMSPI (X'02')	 — — SPI	 Bit mask for syncpoint manager Bit mask for application program 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 47.)

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.

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.

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 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 42.
- 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 42. For further details of the DFHEIENT and DFHEIRET macros, see DFHECALL macro.
- 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 Transferring data between programs using channels.

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.

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 5. 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 local work area are addressed by parameters UEPTAA and UEPTAL of the DFHUEPAR parameter list.

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 24.
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 4 on page 38 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 24.)

Table 4. 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

Table 4. Valid return codes for a TRUE invoked by the CICS syncpoint manager (continued)		
Request-type	**Return codes**	**Meaning**
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 5. 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 38, 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 47.

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.

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 26. 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 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 27). 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 6 on page 44. For further information about coding a DFHEIENT call, see DFHECALL macro.

Sample code for a TRUE invoked at CICS termination

Note that the sample in Figure 6 on page 44 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 6. 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 7. 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 8 on page 45.

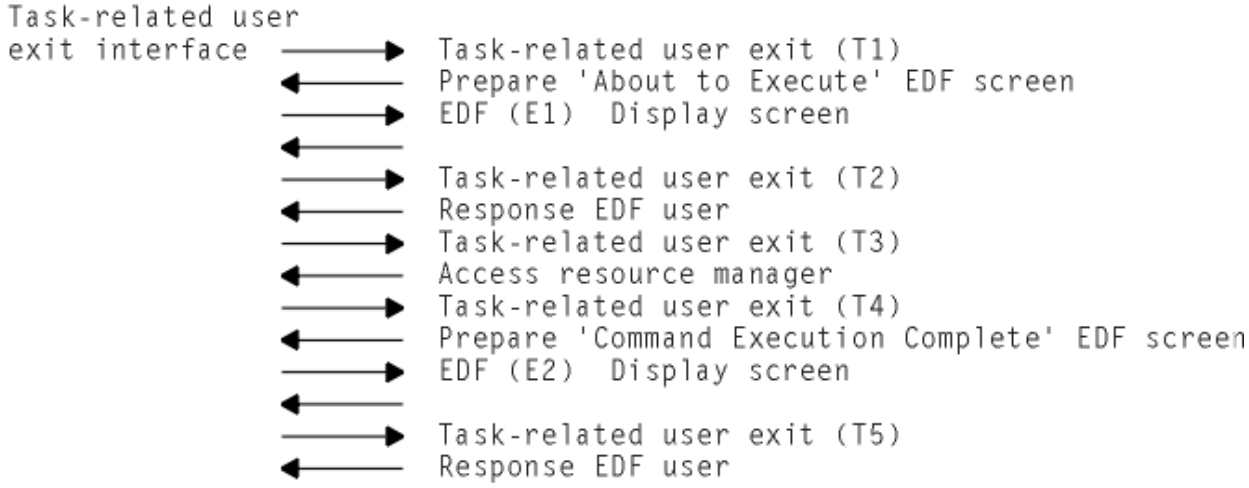


Figure 8. Interface between the task-related user exit and EDF

Table 5 on page 46 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 8 on page 45.

Table 5. 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 main 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.

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, see Introduction to System programming commands.

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 that is issued by the application program.

Note: If adapter data has already been set, then the DFH$APDT sample does not attempt to change it.

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.

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.

XPI functions are listed and described in detail in XPI functions (by domain).

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 the user exit to succeed, you must also check other factors, for example whether XPI parameters have changed between releases. For details, see Changes to CICS XPI.

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 59.

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 58), 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 59.

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 59).

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 timeout 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 54. 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 Global user exit sample program DFH$XTSE.

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 59.

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 the XPI.

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.

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 58.

```
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 9. 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 10. 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 11. 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 12. 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 49. 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 2. Customizing with initialization and shutdown programs

You can write programs to run during the initialization and shutdown phases of CICS processing.

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).

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(Any)
  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:

- Attempting to access any external resource during PLT processing might fail depending on the state of the inter-system connection.
- 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. PLTPI programs can issue **EXEC CICS RUN TRANSID**, **EXEC CICS FETCH CHILD** and **EXEC CICS FETCH ANY** commands. The issuing task could be suspended if the response from the child task is not yet available, which may affect the time at which control is given to CICS.
- 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 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 Defining runtime options for Language Environment.
- PLTPI programs must not use the **EXEC CICS INVOKE SERVICE** command. At this stage of CICS initialization, pipeline scans have not started, so web service requests will fail because the WEBSERVICE definitions, created as a result of the pipeline scan, have not been created yet.
- 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(ANY) 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 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).

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 z/Architecture Principles of Operation. 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.
- 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.

Chapter 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 CICSTS55.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, see 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.

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 storage 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.

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 13 on page 71. 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 “Assembling and link-editing user-replaceable programs” on page 330.

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, GCDSA or GUDSA), 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, the floating point registers, and the vector 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 13 on page 71 shows the assembler language source code of the default program error program. Figure 14 on page 72 through Figure 16 on page 74 show the source code of 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 13. Source code of the default program error program (DFHPEP)

Figure 14 on page 72 through Figure 16 on page 74 show the assembler language source code of the communication area of the default program error program.

```
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
*
```

Figure 14. Source of DFHPEP communication area (assembler-language)

```
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
PEP_COM_FLAG1                 DS     X         Flag byte
PEP_COM_GP64_REGS_AVAIL       EQU    X'80'     64 bit register values
*                                              available in
*                                              PEP_COM_G64_REGISTERS
PEP_COM_ACCESS_REGS_AVAIL     EQU    X'40'     64 bit register values
*                                              available in
*                                              PEP_COM_ACCESS_REGISTERS
PEP_COM_ORIGINAL_FPR_AVAIL    EQU    X'20'     FPR 0, 2, 4 & 6 values
*                                              available in
*                                              PEP_COM_FP_REGISTERS
PEP_COM_ADDITIONAL_FPR_AVAIL  EQU    X'10'     All FPR available in
*                                              PEP_COM_FP_REGISTERS &
*                                              FPCR in
*                                              PEP_COM_FPC_REGISTER
                              DS     CL7       Reserved
PEP_COM_GP64_REGISTERS        DS     CL128     64 bit GP registers
PEP_COM_FP_REGISTERS          DS     0CL132    FP registers
PEP_COM_FP_REGISTER0          DS     FD        FP register 0
PEP_COM_FP_REGISTER1          DS     FD        FP register 1
PEP_COM_FP_REGISTER2          DS     FD        FP register 2
PEP_COM_FP_REGISTER3          DS     FD        FP register 3
PEP_COM_FP_REGISTER4          DS     FD        FP register 4
PEP_COM_FP_REGISTER5          DS     FD        FP register 5
PEP_COM_FP_REGISTER6          DS     FD        FP register 6
PEP_COM_FP_REGISTER7          DS     FD        FP register 7
PEP_COM_FP_REGISTER8          DS     FD        FP register 8
PEP_COM_FP_REGISTER9          DS     FD        FP register 9
PEP_COM_FP_REGISTER10         DS     FD        FP register 10
PEP_COM_FP_REGISTER11         DS     FD        FP register 11
PEP_COM_FP_REGISTER12         DS     FD        FP register 12
PEP_COM_FP_REGISTER13         DS     FD        FP register 13
PEP_COM_FP_REGISTER14         DS     FD        FP register 14
PEP_COM_FP_REGISTER14         DS     FD        FP register 15
PEP_COM_FPC_REGISTER          DS     F         FPC register
PEP_COM_ACCESS_REGISTERS      DS     CL64      Access registers
*
```

Figure 15. Source of DFHPEP communication area (assembler-language) continued

```
*
*                               16 byte PSW at time of abend
*
PEP_COM_PSW16                   DS     CL16      16 byte PSW
*
*                               Vector Register Information
*
PEP_COM_VR_REGISTERS            DS     0CL512    VR registers
PEP_COM_VR_REGISTER0            DS     CL16      VR Register 0
PEP_COM_VR_REGISTER1            DS     CL16      VR Register 1
PEP_COM_VR_REGISTER2            DS     CL16      VR Register 2
PEP_COM_VR_REGISTER3            DS     CL16      VR Register 3
PEP_COM_VR_REGISTER4            DS     CL16      VR Register 4
PEP_COM_VR_REGISTER5            DS     CL16      VR Register 5
PEP_COM_VR_REGISTER6            DS     CL16      VR Register 6
PEP_COM_VR_REGISTER7            DS     CL16      VR Register 7
PEP_COM_VR_REGISTER8            DS     CL16      VR Register 8
PEP_COM_VR_REGISTER9            DS     CL16      VR Register 9
PEP_COM_VR_REGISTER10           DS     CL16      VR Register 10
PEP_COM_VR_REGISTER11           DS     CL16      VR Register 11
PEP_COM_VR_REGISTER12           DS     CL16      VR Register 12
PEP_COM_VR_REGISTER13           DS     CL16      VR Register 13
PEP_COM_VR_REGISTER14           DS     CL16      VR Register 14
PEP_COM_VR_REGISTER15           DS     CL16      VR Register 15
PEP_COM_VR_REGISTER16           DS     CL16      VR Register 16
PEP_COM_VR_REGISTER17           DS     CL16      VR Register 17
PEP_COM_VR_REGISTER18           DS     CL16      VR Register 18
PEP_COM_VR_REGISTER19           DS     CL16      VR Register 19
PEP_COM_VR_REGISTER20           DS     CL16      VR Register 20
PEP_COM_VR_REGISTER21           DS     CL16      VR Register 21
PEP_COM_VR_REGISTER22           DS     CL16      VR Register 22
PEP_COM_VR_REGISTER23           DS     CL16      VR Register 23
PEP_COM_VR_REGISTER24           DS     CL16      VR Register 24
PEP_COM_VR_REGISTER25           DS     CL16      VR Register 25
PEP_COM_VR_REGISTER26           DS     CL16      VR Register 26
PEP_COM_VR_REGISTER27           DS     CL16      VR Register 27
PEP_COM_VR_REGISTER28           DS     CL16      VR Register 28
PEP_COM_VR_REGISTER29           DS     CL16      VR Register 29
PEP_COM_VR_REGISTER30           DS     CL16      VR Register 30
PEP_COM_VR_REGISTER31           DS     CL16      VR Register 31

*                               length of DFHPEP_COMMAREA
*
PEP_COM_LEN EQU *-PEP_COM_STANDARD
```

Figure 16. Source of DFHPEP communication area (assembler-language) continued

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 CICSTS55.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 CICSTS55.CICS.SDFHMAC and CICSTS55.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.

Writing a custom EP adapter

A custom EP adapter is a CICS program associated with an event binding that formats and then emits events produced by the event binding.

About this task

If the CICS-supplied EP adapters do not meet your requirements for processing events, you can write your own custom EP adapter to process the event data yourself.

CICS invokes the EP adapter for each event that is emitted. The input to a custom EP adapter is the current channel, which contains the CICS event object as a collection of containers. The containers are: DFHEP.CONTEXT, DFHEP.DESCRIPTOR, DFHEP.ADAPTER, DFHEP.ADAPTPARM, DFHEP.CHAR.*nnnnn*,

DFHEP.DATA.*nnnnn*, and DFHEP.ERR.*nnnnn*. Copybooks are provided for the DFHEP.CONTEXT, DFHEP.DESCRIPTOR, and DFHEP.ADAPTPARM containers. These copybooks can change between releases of CICS; therefore, you should recompile custom EP adapters for each new CICS release.

In addition to the emitted event, the custom EP adapter must produce an indication of success or failure.

Procedure

1. Include the event context data copybook for your programming language.

   This copybook describes the DFHEP.CONTEXT container of context data for the event your EP adapter is processing.

   - DFHEPCXD for Assembler language
   - DFHEPCX0 for COBOL
   - DFHEPCXL for PL/I
   - DFHEPCXH for C
2. Include the event descriptor copybook for your programming language.

   This copybook describes the DFHEP.DESCRIPTOR container that describes the captured business data for the event your EP adapter is processing.

   - DFHEPDED for Assembler language
   - DFHEPDE0 for COBOL
   - DFHEPDEL for PL/I
   - DFHEPDEH for C
3. Include the EP adapter parameters copybook for your programming language.

   This copybook describes the DFHEP.ADAPTPARM container.

   - DFHEPAPD for Assembler language
   - DFHEPAP0 for COBOL
   - DFHEPAPL for PL/I
   - DFHEPAPH for C
4. Get the context information from the DFHEP.CONTEXT container using the event context data copybook.
5. Get the EP adapter custom data. The DFHEP.ADAPTER container contains the data that is specified in the field **Data passed to the Custom Adapter** that is in the **Adapter** tab of the event binding editor for your event binding.

   Note: The data in this container is restricted to EBCDIC characters.
6. Get the EP adapter parameters from the DFHEP.ADAPTPARM container using the EP adapter parameters configuration copy book.

   An important item of information in the DFHEP.ADAPTPARM container is the emission recoverability indicator, EPAP-RECOVER. This container also includes the adapter name, EPAP-ADAPTER-NAME.
7. The custom adapter must verify the EPAP-RECOVER setting in the DFHEP.ADAPTPARM container in the custom EP adapter.

   EP adapters must emit the events by using a transport in a recoverable or unrecoverable way. If the setting is not EPAP-ANY-RECOVERABLE, you must honor the EPAP-RECOVER setting. To support synchronous emission, EP adapters must be sensitive to the transport recoverability requirement that is indicated by the EPAP-RECOVER setting in the context container:

   - If the field is set to EPAP-RECOVERABLE, the EP adapter must write to the transport in a recoverable way.
   - If the field is set to EPAP-NON-RECOVERABLE, the EP adapter must ***not*** write to the transport in a recoverable way.

You must terminate the adapter if you cannot honor the recoverability setting; otherwise, the transactional requirement for the event is not implemented correctly.

The field is set to EPAP-ANY-RECOVERABLE for asynchronous emission.

8. Get the event descriptor from the DFHEP.DESCRIPTOR container using the event descriptor copybook.

   The event descriptor consists of a prefix describing the number of data items in the descriptor and a descriptor for each data item. The data item descriptor includes the name of the business data item and its type. The data items themselves are in containers with a name of format DFHEP.CHAR.*nnnnn* and DFHEP.DATA.*nnnnn*, where *nnnnn* is a 5-digit sequence number that indicates the ordering of the captured data starting at 00001. The DFHEP.CHAR.*nnnnn* containers contain the capture data in a printable (character) form formatted as requested in the event specification. The DFHEP.DATA.*nnnnn* containers contain the unformatted capture data.

9. Get the required data items from the DFHEP.DATA.*nnnnn* containers. Alternatively, if you do not wish to format the data yourself, the data formatted as requested in the event specification can be obtained from the corresponding DFHEP.CHAR.*nnnnn* container.

   Note: If a data item was not available for data capture, the corresponding DFHEP.DATA.*nnnnn* container is not present in the CICS event object. This situation can happen, for example, when a capture specification specifies a capture data item associated with an optional parameter that was not present on the API command that caused the event. Instead the DFHEP.ERR.*nnnnn* container will be present in the CICS event object and contains the capture length specified for the capture data item. The corresponding DFHEP.CHAR.*nnnnn* container will be present in the CICS event object but will contain one or more asterisks.

10. Format the data and emit the event.

    Each item in the DESCRIPTOR array defines the type of the source data captured and the required length and type of that data when and if it is formatted.

    Data is captured up to the length specified in the capture data item spec. The captured data is exactly as found in the source data areas. If the capture data is not available then the corresponding DFHEP.DATA container is not created.

    A format length of **Automatic** (0) is supported for all data types. When a format length of **Automatic** is used, the equivalent EPDE field is set to epde_formatLen_auto.

    A format precision of **Automatic**, resulting in the length that is required for the specified data type and precision being used, is supported for all numeric data types. When a format precision of **Automatic** is used, the equivalent EPDE field is set to epde_formatPrec_auto.

11. Complete processing by issuing an **EXEC CICS RETURN** or **EXEC CICS ABEND** command.

    In addition to the emitted event, the custom EP adapter must produce an indication of success or failure. A custom EP adapter that cannot emit the event must end with an abend code so that CICS can start any required recovery actions and increment the appropriate statistics.

An example code fragment in the COBOL language

This code fragment, taken from the sample custom EP adapter DFH0EPAC, shows the sequence of steps described in this procedure. It does not include any processing of the EP adapter information or the data items. The complete sample can be found in the CICSTS 55 .CICS.SDFHSAMP library.

```
****************************************************************
 Linkage section.
****************************************************************
 01 EPContext.
    copy dfhepcxo.
 01 EPDescriptor.
    copy dfhepdeo.
 01 EPAdapter          pic x(16).
 01 EPAdaptparm
    copy dfhepapo.
 01 EPData             pic x(32000).
****************************************************************
```

```
 Main-program section.
*****************************************************************
*
     perform Initial-processing.
*
*    Process the data items
     perform Process-data-item
             varying ItemNum from 1 by 1
             until ItemNum > epde-itemcount.
*
*****************************************************************
* Any final EVENT PROCESSING code to go here
*****************************************************************
*
*    Return to caller
     EXEC CICS RETURN END-EXEC.
*
 Main-program-exit.
     exit.
*
*****************************************************************
 Initial-processing section.
*****************************************************************
*
*    Obtain the DFHEP.CONTEXT container
     EXEC CICS GET CONTAINER('DFHEP.CONTEXT')
                   SET(address of EPContext)
                   FLENGTH(EPContextLength)
     END-EXEC.
*
*    Obtain the DFHEP.DESCRIPTOR container
     EXEC CICS GET CONTAINER('DFHEP.DESCRIPTOR')
                   SET(address of EPDescriptor)
                   FLENGTH(EPDescriptorLength)
     END-EXEC.
*
*    Obtain the DFHEP.ADAPTER container
     EXEC CICS GET CONTAINER('DFHEP.ADAPTER')
                   SET(address of EPAdapter)
                   FLENGTH(EPAdapterLength)
     END-EXEC.
*
*    Obtain the DFHEP.ADAPTPARM container
     EXEC CICS GET CONTAINER('DFHEP.ADAPTPARM')
                   SET(address of EPAdaptparm)
                   FLENGTH(EPAdaptparmLength)
     END-EXEC.
*

*    Check the recoverability of the transport is right for the event
     if not epap-any-recoverable
       perform Check-recoverability.

*
 Initial-processing-exit.
     exit.
*
*****************************************************************
 Process-data-item section.
*****************************************************************
*
*    Process a data descriptor item
*
*    Build the data container name: DFHEP.DATA.nnnnn
     string 'DFHEP.DATA.' delimited by size
            ItemNum         delimited by size
            into ContainerName
     end-string.
*
*    Obtain the DFHEP.DATA.nnnnn container - if present
     EXEC CICS GET CONTAINER(ContainerName)
                   SET(address of EPData)
                   FLENGTH(EPDataLength)
                   RESP(Resp) RESP2(Resp2)
     END-EXEC.
*****************************************************************
* Any final code to process DATA ITEM to go here
*****************************************************************
*
*    Convert the data according to epde-datatype
     perform Convert-data.
*
```

```
*    Calculate the target field length
     perform Calculate-length.
*
*    Format the data according to epde-formattype
     perform Format-data.
*
*    Move over the data item ready for the next one
     add TSQFieldLength to TSQFieldIndex.
*
 Process-data-item-exit.
     exit.
*
****************************************************************
```

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.

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 CICSTS55.CICS.SDFHSAMP library.

The assembler copybook for mapping the communications area is in the CICSTS55.CICS.SDFHMAC library.

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 "Writing a node error program" on page 106.

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 99.

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.

A format description of the terminal abnormal condition line entry (TACLE) DSECT is provided under “Writing a terminal error program” on page 99.

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 “Assembling and link-editing user-replaceable programs” on page 330.

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 17 on page 83):

- 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.

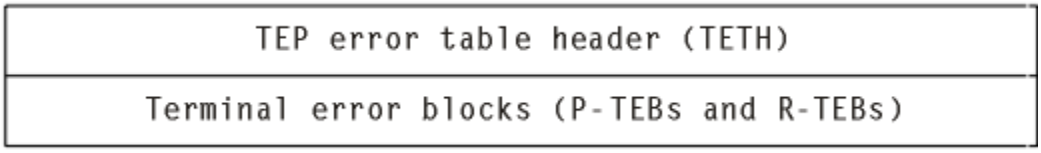


Figure 17. TEP error table

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 18 on page 83.

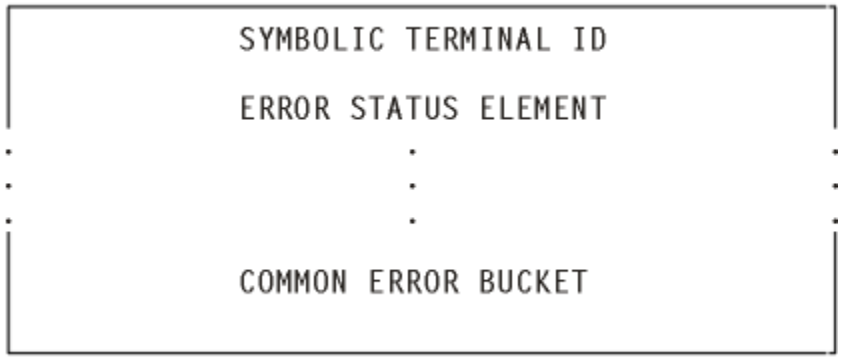


Figure 18. Terminal error block (TEB)

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 19 on page 87 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.

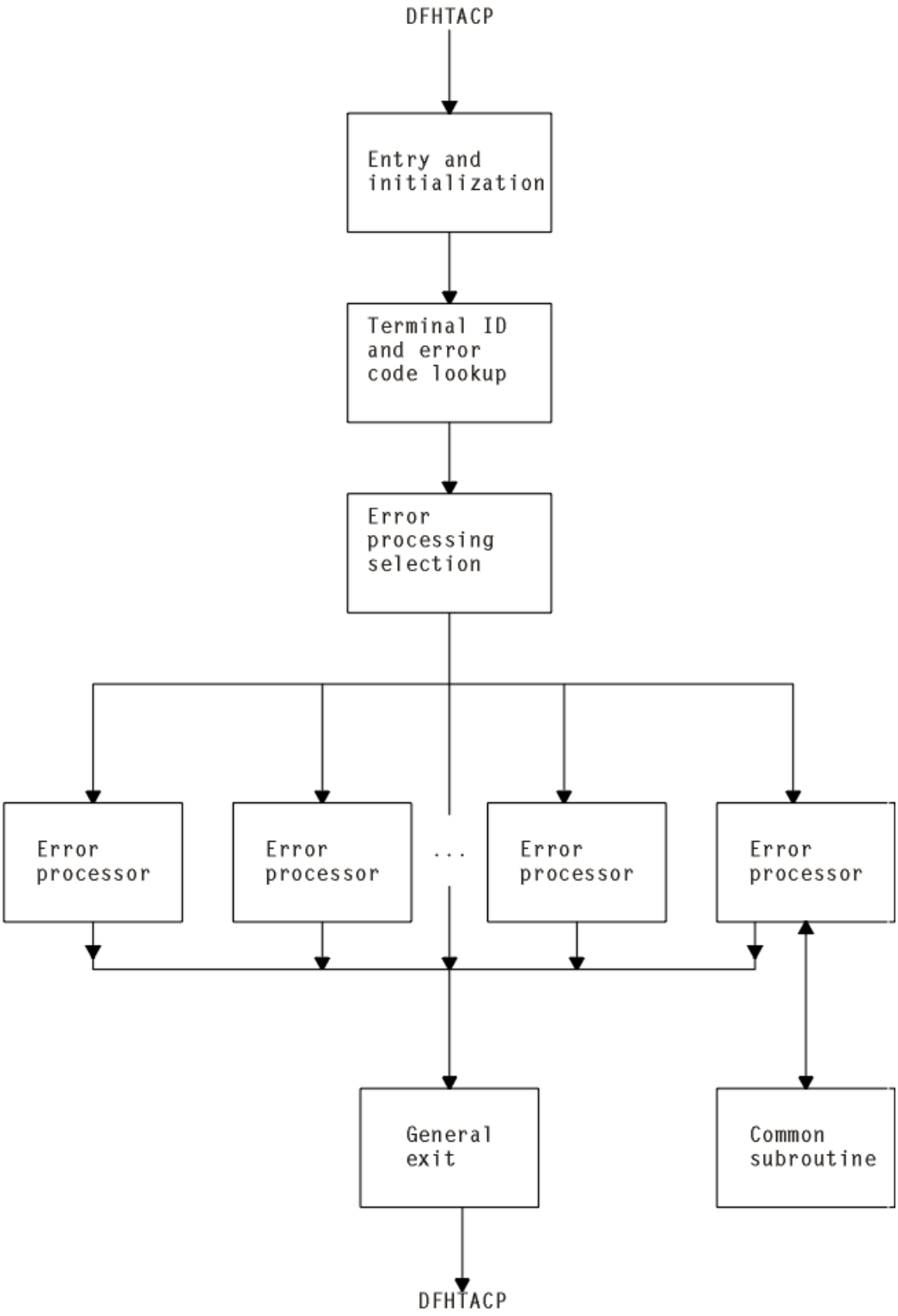


Figure 19. Overview of the sample terminal error program (DFHXTEP)

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 100.

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 99. 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 99. 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 6. 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 7. 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 "Assembling and link-editing user-replaceable programs" on page 330.

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 "Assembling and link-editing user-replaceable programs" on page 330.

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 23 on page 103.

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 20 on page 94 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 20. 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 8 on page 97.

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 23 on page 103. 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 8 on page 97). 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 8 on page 97). 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 8 on page 97 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 8. Default thresholds of the sample TEP

CODE=	COUNT=	TIME=
81	3	(7,MIN)
84	1	0
85	1	0
87	50 (Note “2” on page 97)	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” on page 97)	0	0
96	2	(1,MIN)
97 (Note “1” on page 97)	0	0
99	1	0
9F (Note “1” on page 97)	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 21 on page 98 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 21. 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 9 on page 99.

Table 9. Supplied source files and macros

Name	Type	Description	Library
DFHXTEP	Source	Sample terminal error program (assembler language)	CICSTS55.CICS.SDFHSAMP
DFHXTEPT	CSECT	Sample terminal error tables (assembler language)	CICSTS55.CICS.SDFHSAMP
DFHTEPM	Macro	Sample TEP program generator (assembler language)	CICSTS55.CICS.SDFHMAC
DFHTEPT	Macro	TEP table generator (assembler language)	CICSTS55.CICS.SDFHMAC
DFHTEPCA	Macro	assembler language communication area	CICSTS55.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 81. 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 22 on page 100.

```
                              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 22. 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 page 101).

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 23 on page 103.

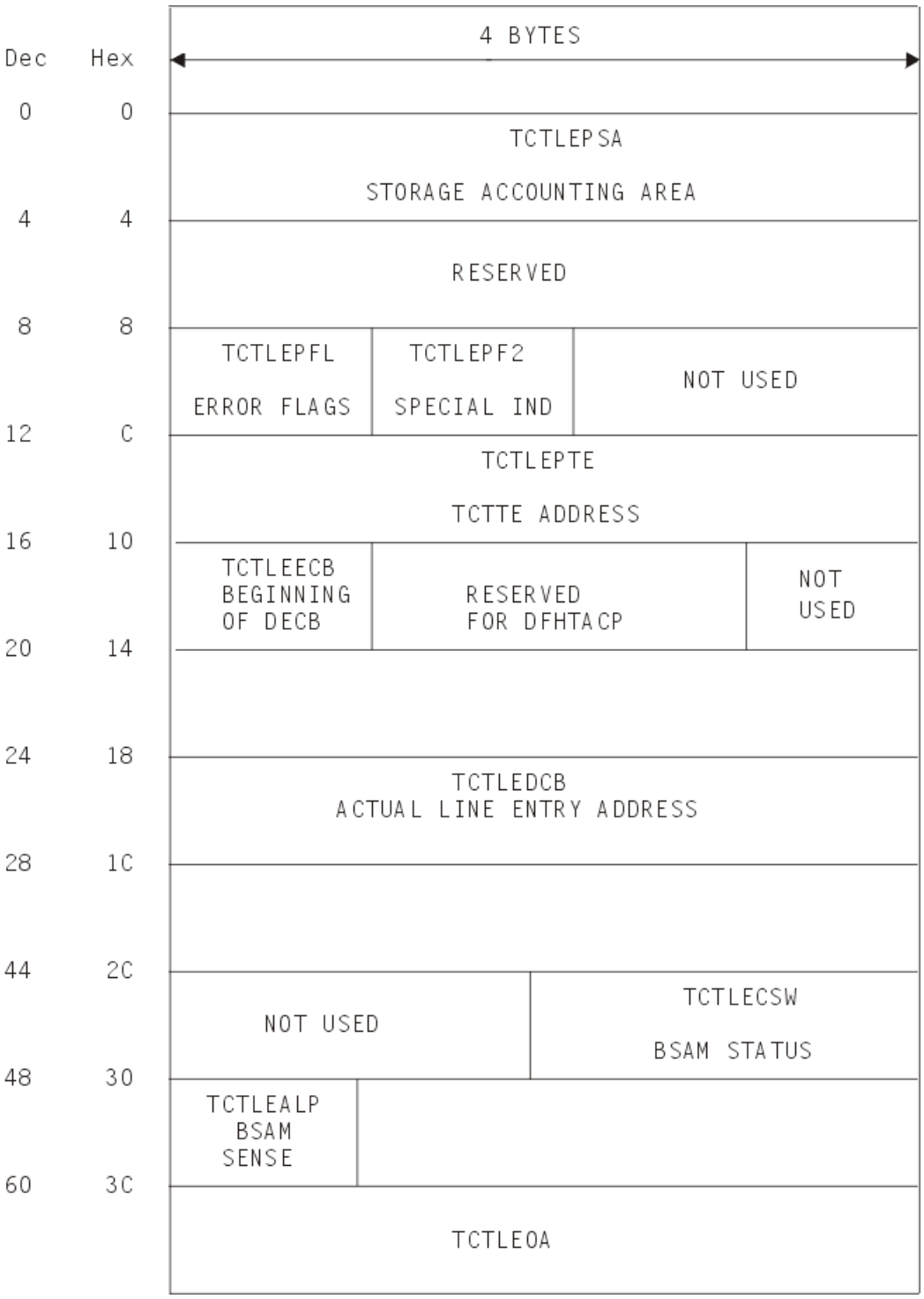


Figure 23. Format description of the TACLE DSECT: TERMINAL ABNORMAL CONDITION LINE ENTRY

```
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 24. Text description of the format of the TACLE DSECT: TERMINAL ABNORMAL CONDITION LINE ENTRY

Example of a user-written terminal error program

The "DFHTEP recursive retry routine" on page 105 is an example of the logic steps necessary to design a portion of the terminal error program. In Figure 25 on page 105, 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 25. 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 26. DFHTEP recursive retry routine (part 2)

Note that the code in Figure 25 on page 105 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.

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 112. 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 Default actions of the node abnormal condition program.)

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 113.

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 Default actions of the node abnormal condition program.

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 116) 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 120.

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 124).

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 121 and "DFHSNET—generating the node error table" on page 126. 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 123.

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 124, 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 123.) 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 123.

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 112.

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 131) 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 131.) 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 Default actions of the node abnormal condition program.

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 27 on page 113 illustrates the general structure of the communication area.

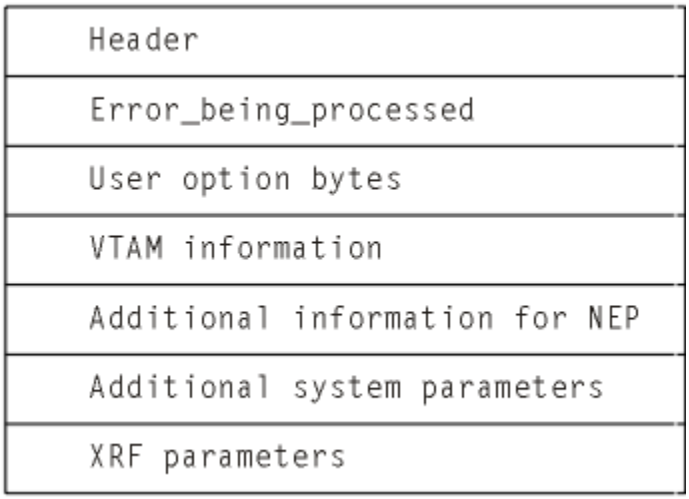


Figure 27. General structure of the communication area

The significance of each section of the communication area is described here:

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 28 on page 114.

```
*****************************************************************************
**                       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 28. 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 29. 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 30. 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 129, 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 intended 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

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.

[1] Do not set this flag when processing error code X'49' (TCZCLSIN).

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 123.

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. 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 “Writing a terminal error program” on page 80.

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.

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.

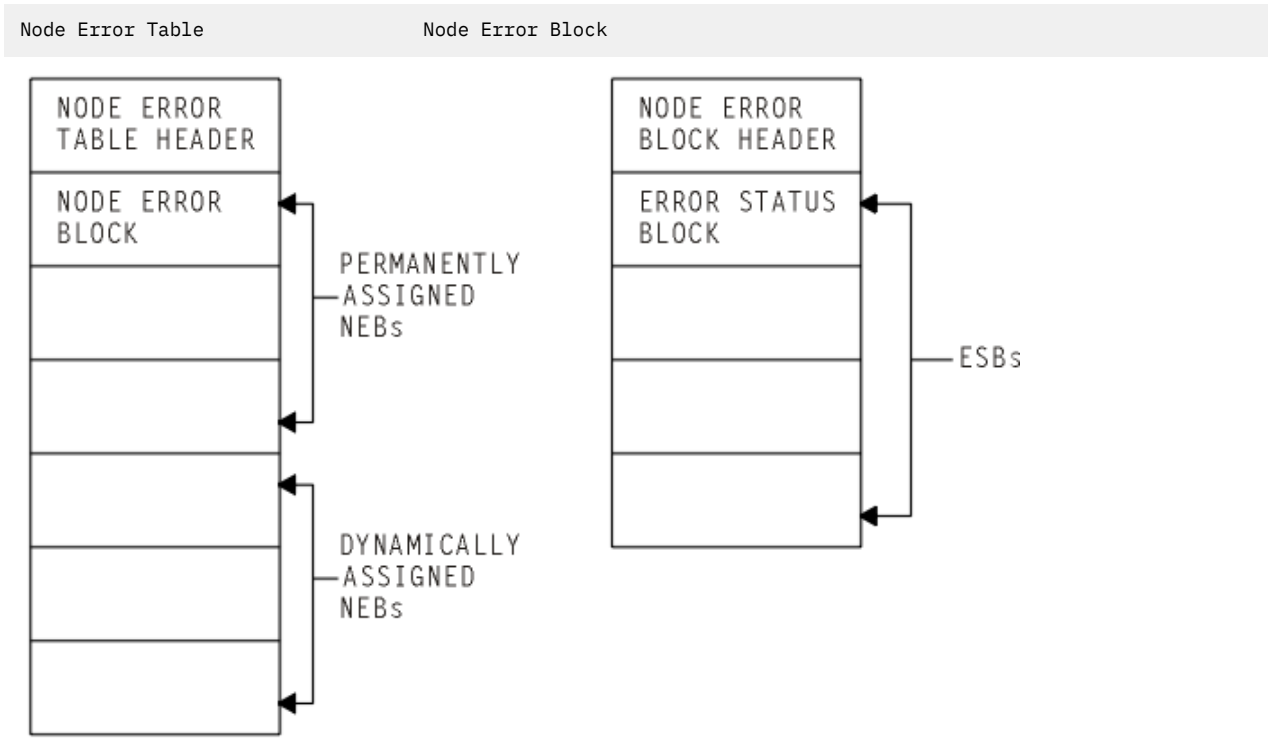


Figure 31. Format of node error table and node error block

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 129.)

Optional error processor for interactive logical units

One 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 "Assembling and link-editing user-replaceable programs" on page 330.

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.

DFHSNEP TYPE=USTOR

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.

DFHSNEP TYPE=USTOREND

The DFHSNEP TYPE=USTOREND macro has the following format:

```
DFHSNEP  TYPE=USTOREND
```

This macro indicates the end of user storage definitions. Its use is mandatory if 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 28 on page 114.
- The fact that certain functions cannot be used within DFHZNEP. (See “Restrictions on the use of EXEC CICS commands” on page 129.)
- The register conventions used by the sample node error program. These are described in Table 10 on page 125.

Table 10. Register assignment

Register	Use
0	Work register
1	Address of the EXEC parameter list

Table 10. Register assignment (continued)	
Register	**Use**
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 11 on page 128.

Table 11. Supplied source files, copy books, and macros

Name	Type	Description	Library
DFHZNEP0	Program	Default node error program (assembler language)	CICSTS55.CICS.SDFHSAMP
DFHZNEPX	Source	Default NEP (embedded by DFHZNEP0 via COPY statement)	CICSTS55.CICS.SDFHSAMP
DFHSNEP	Macro	Sample NEP program generator (assembler language)	CICSTS55.CICS.SDFHMAC
DFHZNEPI	Macro	NEP interface generator (for multiple NEPs)	CICSTS55.CICS.SDFHMAC
DFHNEPCA	Macro	assembler language communication area	CICSTS55.CICS.SDFHMAC
DFHNEPCA	Copy book	C-language communication area	CICSTS55.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 28 on page 114) 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 129.

Coding for specific SNA sense codes

About this task

Figure 32 on page 131 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 32. 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 111. 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 and TCSACTN system initialization parameter).

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 TYPETERM attributes. 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.

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 “Writing a program to control autoinstall of APPC connections” on page 160. For information about controlling the installation of shipped LUs and connections, see “Writing a program to control autoinstall of shipped terminals” on page 173. For information about controlling the installation of virtual LUs used by the CICS Client products and the 3270 bridge , see “Writing a program to control autoinstall of virtual terminals” on page 179.

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 Autoinstall. 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 "Writing a program to control autoinstall of APPC connections" on page 160 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 Coding entries in the VTAM LOGON mode table 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 149.

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 156. 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 163. The parameter list passed at INSTALL of shipped terminals and connections is described in “The communications area at INSTALL for shipped terminals” on page 176. The parameter list passed at INSTALL of client virtual terminals is described in “The communications area at INSTALL for Client virtual terminals” on page 183. The parameter list passed at INSTALL of MVS consoles is described in “Writing a program to control autoinstall of consoles” on page 155. 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 33 on page 137.

```
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 33. 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.

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 34 on page 139 shows all of these fields in their required order.

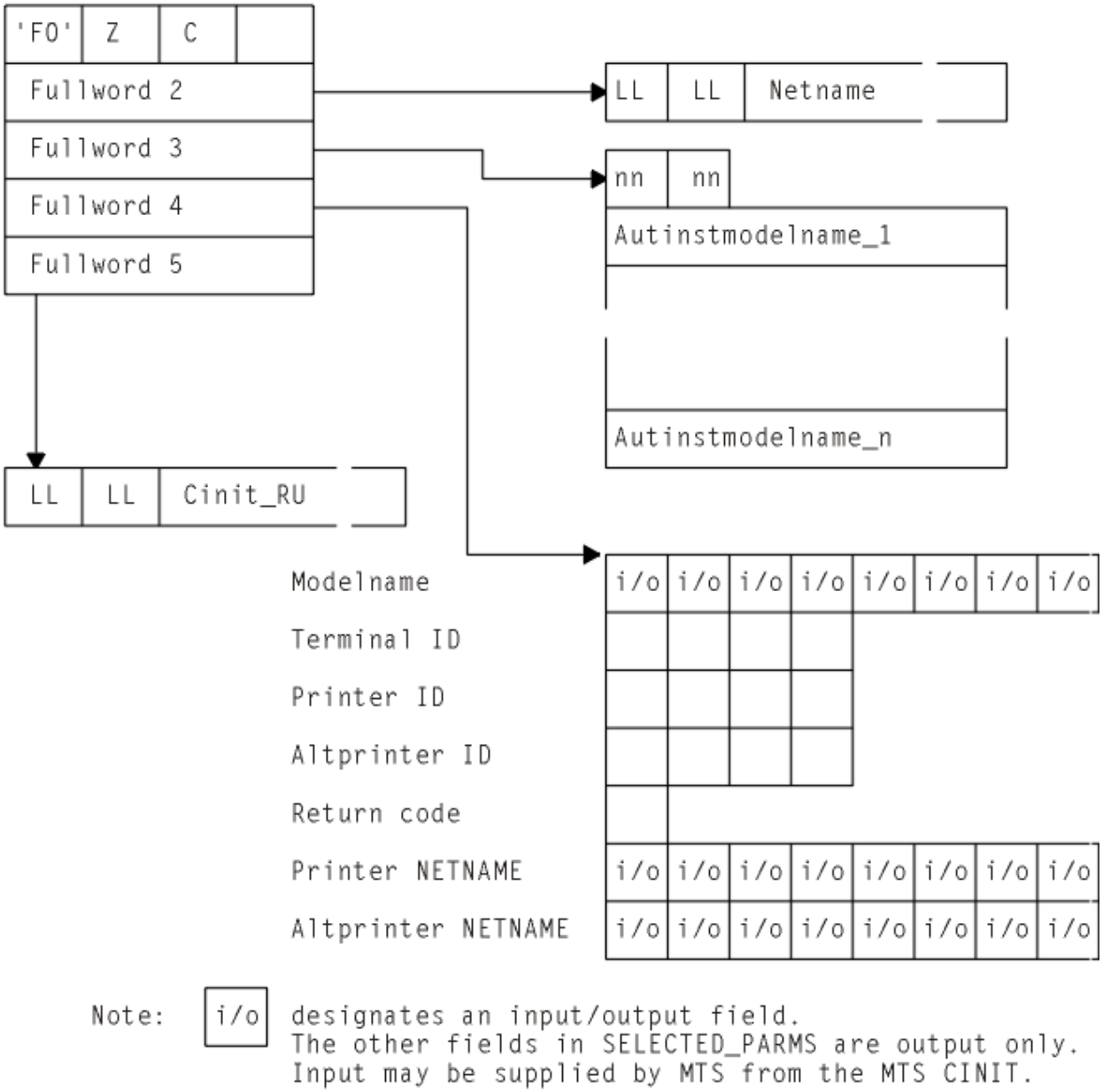


Figure 34. Autoinstall control program's parameter list at INSTALL

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. Autoinstall models describes the definition of models.

TYPETERM device types and pointers to related LOGON mode data 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 142.

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 141 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 CONNECTION attributes. 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.

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 TERMINAL attributes.

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 141.)

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 143 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 35 on page 144.

```
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 35. Autoinstall control program's communication area at DELETE

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 165. 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 178. A value of X'FC' represents the deletion of a client virtual terminal; see “The autoinstall control program at DELETE” on page 185.
2. The terminal identifier of the deleted resource, as shown in Table 12 on page 144.

Table 12. 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.
- The **EXEC CICS SET AUTOINSTALL** command or the **CEMT SET AUTOINSTALL** command. For more information about these commands, see SET AUTOINSTALL and CEMT SET AUTOINSTALL.

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 137. 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.

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.

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 36 on page 146. 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 36 on page 146 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 36. 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 CICSTS55.CICS libraries in which they can be found, are summarized in Table 13 on page 147.

Table 13. 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

Table 13. Sample "good night" program, copy books, and mapset (continued)		
Language	**Member name**	**Library**
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.

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 CICSTS55.CICS libraries in which they can be found, are summarized in Table 14 on page 149. Note that the COBOL, PL/I, and C copy books each have an alias of DFHTCUDS.

Table 14. 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 CICSTS55.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 CICSTS55.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 application programs. 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" on page 150. 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 37 on page 151, 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 37. 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 38. Example of how to customize the DFHZATDX sample program (part 2)

COBOL

Figure 39 on page 153, 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 39. Example of how to customize the DFHZCTDX sample program

PL/I

Figure 40 on page 154, 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 40. 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 41. Example of how to customize the DFHZPTDX sample program (part 2)

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 “Writing a program to control autoinstall of LUs” on page 135.

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 135).

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 137. 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 installation 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 42 on page 157.

```
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 42. 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 43 on page 158 shows all of these fields in their required order.

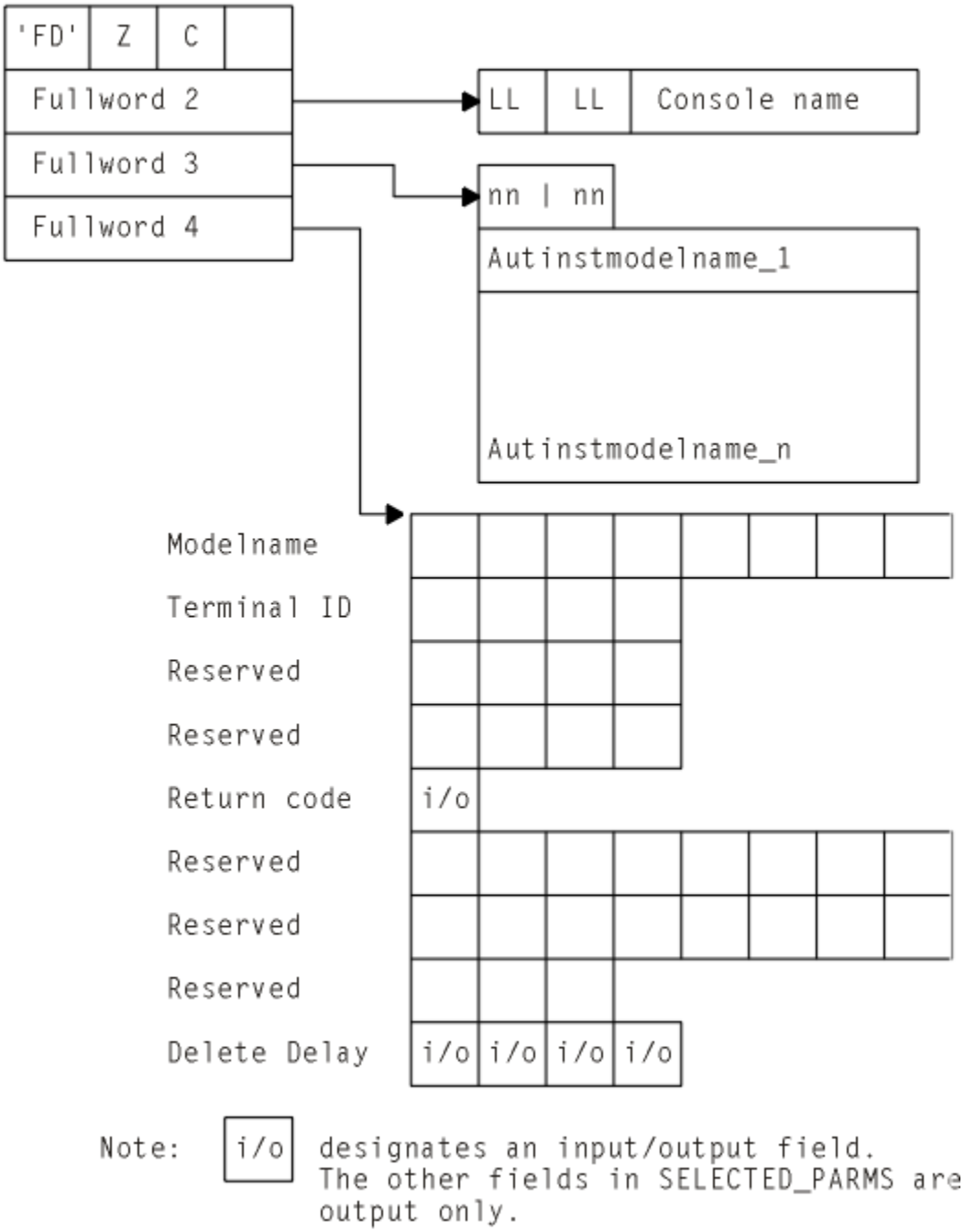


Figure 43. Autoinstall control program's parameter list at INSTALL

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 44. 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 149.

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 “Writing a program to control autoinstall of LUs” on page 135. For information about controlling the installation of shipped terminals and connections, see “Writing a program to control autoinstall of shipped terminals” on page

173. For information about controlling the installation of virtual LUs used by CICS clients, see “Writing a program to control autoinstall of virtual terminals” on page 179.

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 “Writing a program to control autoinstall of LUs” on page 135.

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 “Writing a program to control autoinstall of LUs” on page 135. 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 137. The parameter list passed at INSTALL of MVS consoles is described in “Autoinstall control program at INSTALL” on page 156. The parameter list passed at INSTALL of shipped terminals and connections is described in “The communications area at INSTALL for shipped terminals” on page 176. The parameter list passed at INSTALL of Client virtual terminals is described in “The communications area

at INSTALL for Client virtual terminals” on page 183. 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 CICSTS55.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 45. Autoinstall control program's communications area at INSTALL

INSTALL_APPC_STANDARD header
A fullword input field comprising the following information:

INSTALL_APPC_EXIT_FUNCTION
A 1-byte field that defines the installation 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 5 .

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 “Writing a program to control autoinstall of LUs” on page 135.

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 Configuring z/OS Communications Server generic resources in Configuring.)

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 5 .

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 Defining connections in a generic resource environment.)

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 46 on page 165.

```
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 46. Autoinstall control program's communication area at DELETE

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 143. 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 178. The value X'FC' represents the deletion of a Client virtual terminal. For more information, see "The autoinstall control program at DELETE" on page 185.

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).

- 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.

  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 CICSTS55.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 "Writing a program to control autoinstall of LUs" on page 135. 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 “Writing a program to control autoinstall of LUs” on page 135.

   X'F1'
   A delete request for a terminal or APPC single-session device. Proceed as for DFHZATDX. See “Writing a program to control autoinstall of LUs” on page 135.

   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 5 .

   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)
```

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.

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.

The SENDCOUNT attribute on the autoinstalled IPCONN resource is set to the same value as RECEIVECOUNT, or the SENDCOUNT 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 172.

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 CICSTS55.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 47. Autoinstall user program's communications area at INSTALL

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 48 on page 171.

```
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 48. Autoinstall user program's communication area at DELETE

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 it moves 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 CICSTS55.CICS libraries in which they can be found, are summarized in Table 15 on page 172.

Table 15. 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
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.

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 Local and remote names for terminals.

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" on page 176.

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 137. The parameter list passed at INSTALL of MVS consoles is described in "Autoinstall control program at INSTALL" on page 156. 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 163. The parameter list passed at INSTALL of Client virtual terminals is described in "The communications area at INSTALL for Client virtual terminals" on page 183. 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 CICSTS55.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 49. 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 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 TERMINAL attributes.

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 174.

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 Efficient deletion of shipped terminal definitions.

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 175.

The parameter list passed to your user program at DELETE of local terminals is described in "The autoinstall control program at DELETE" on page 143. The parameter list passed at DELETE of local APPC connections is described in "The autoinstall control program at DELETE" on page 165. The parameter list passed at DELETE of Client virtual terminals is described in "The autoinstall control program at DELETE" on page 185. 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 Efficient deletion of shipped terminal definitions.

Figure 50 on page 178 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 50. Autoinstall control program's communications area at DELETE

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

This topic describes the default actions of the supplied autoinstall control programs, DFHZATDX and DFHZATDY when they are invoked at INSTALL and at DELETE.

Default actions at INSTALL

When DFHZATDX or DFHZATDY is invoked at INSTALL of a shipped terminal or connection, it performs the following actions:

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.

Default actions at DELETE

When DFHZATDX or DFHZATDY is invoked at DELETE of a shipped terminal or connection, it takes no action and returns.

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 by using the following sequence.

1. For EPI programs: from the **DevType** parameter of the CICS_EpiAddTerminal function, if this is specified by the Client EPI program.

   For the Client terminal emulator: from the **/m** parameter of the **cicsterm** command used to start the emulator, if this is specified by the workstation user.

   Note: Any autoinstall models specified by Clients must 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 174.

Note: If the supplied name clashes with the name of a local terminal or connection, the installation 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 a 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 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 "Writing a program to control autoinstall of shipped terminals" on page 173.)

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 184 and "The communications area at DELETE for bridge facility virtual terminals " on page 186. 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.

Autoinstall of a Link3270 bridge facility

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 137. 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 163. The parameter list passed at INSTALL of MVS consoles is described in “Autoinstall control program at INSTALL” on page 156. The parameter list passed at INSTALL of shipped terminals and connections is described in “The communications area at INSTALL for shipped terminals” on page 176. This section describes only parameters passed at INSTALL of Client virtual terminals, in “The communications area at INSTALL for Client virtual terminals” on page 183, and of bridge facilities in “The communications area at INSTALL for bridge facility virtual terminals” on page 184.

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 CICSTS55.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 51. 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 TERMINAL attributes.

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 CICSTS55.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 52. 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 TERMINAL attributes. 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 182.

The parameter list passed to your user program at DELETE of local terminals is described in "The autoinstall control program at DELETE" on page 143. The parameter list passed at DELETE of local APPC connections is described in "The autoinstall control program at DELETE" on page 165. The parameter list passed at DELETE of shipped definitions is described in "Autoinstall control program at DELETE" on page 178. This section describes DELETE of Client virtual terminals at "The communications area at DELETE for Client virtual terminals " on page 186 and bridge facility virtual terminals at "The communications area at DELETE for bridge facility virtual terminals " on page 186.

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.

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 53. Communications area of the autoinstall control program at DELETE

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 143. 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 165. A value of X'FA' or X'FB' represents the deletion of a shipped terminal or connection—see Figure 50 on page 178.

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 54. Autoinstall control program's communications area at DELETE

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.

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 188
  - **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 16 on page 189 shows the differences in mapset processing between CICS regions with program autoinstall active and inactive.

Table 16. 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:

- Programs required for Language Environment.
- 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 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 .
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.
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 .
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 CICSTS55.CICS libraries in which they can be found, are summarized in Table 17 on page 194.

Table 17. 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

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 programs 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.

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.
- **EXEC CICS RUN TRANSID** requests.

To route these types of request you must use the distributed routing program. How to write a distributed routing program is described in “Writing a distributed routing program” on page 224.

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.

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 55 on page 197 shows the points at which the dynamic routing program is invoked.

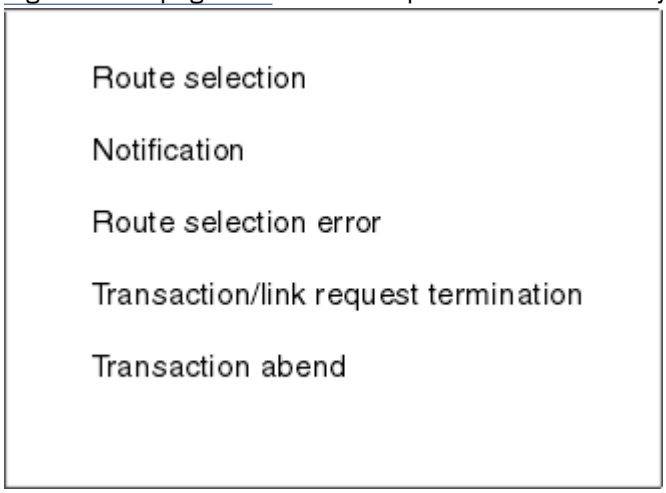


Figure 55. When the dynamic routing program is invoked

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 211. 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 200.

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.

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 56 on page 199 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 56. 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 . For programming information about the INPUTMSG option, see RETURN.

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 211.

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 for z/OS Overview.

  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 211.

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 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 201) 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.
- 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 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 188.

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
- 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. If the return code is not zero, CICS attempts to execute the program in the routing region.

  Special case! Take 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 55 on page 197 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 206.

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. 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.

[2] By “invoked for routing” we mean both “invoked for route selection” and “invoked because an error occurred in route selection”.

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 206.

For information about writing an XPCERES global user exit program, see Program control exits XPCREQ, XPCERES, XPCREQC.

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 211.

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 211.

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.
- 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.
- 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.

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 211. 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 210.

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 210

For information about writing an XPCERES global user exit program, see Program control exits XPCREQ, XPCERES, XPCREQC.

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 Exception conditions for LINK command.

- 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 196)
- Program-link (DPL) requests (described in "Routing DPL requests dynamically" on page 203)
- Non-terminal-related START requests (described in "Routing non-terminal-related START requests" on page 229)

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" on page 211.

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 236 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 for use by CICSPlex System Manager. Currently, it is set as a result of the non-availability of connections to resource managers Db2, IMS, IBM MQ, or VSAM RLS. For more information, see Avoiding the storm drain effect.

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 if no application context is available or when application context is available but application name is not set.

DYRAPPLMAJOR

Application context application major version. It is set to 0 if no application context is available and -1 if the application context is available but application major version is not set.

DYRAPPLMICRO

Application context application micro version. It is set to 0 if no application context is available and -1 if the application context is available but application micro version is not set.

DYRAPPLMINOR

Application context application minor version. It is set to 0 if no application context is available and -1 if the application context is available but application minor version is not set.

DYRAPPLVER

Application context application version. All version numbers set to 0 if no application context is available and -1 if the application context is available but application version is not set.

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 201.

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 201.

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 211 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.

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.

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'
. This value is set only for method requests for enterprise beans and CORBA stateless objects (handled by the distributed routing program).

X'02'
. 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 set to nulls 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 set to nulls if no application context is available or when application context is available but platform name is not set.

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 5 , 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 IBM Knowledge Center.

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 program 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).

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 18 on page 224.

Table 18. 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.

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.
- 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 "Writing a dynamic routing program" on page 196. 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.

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 236.

4. The distributed routing program is invoked at more points than the dynamic routing program. "When the distributed routing program is invoked" on page 229 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 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 57 on page 227 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.

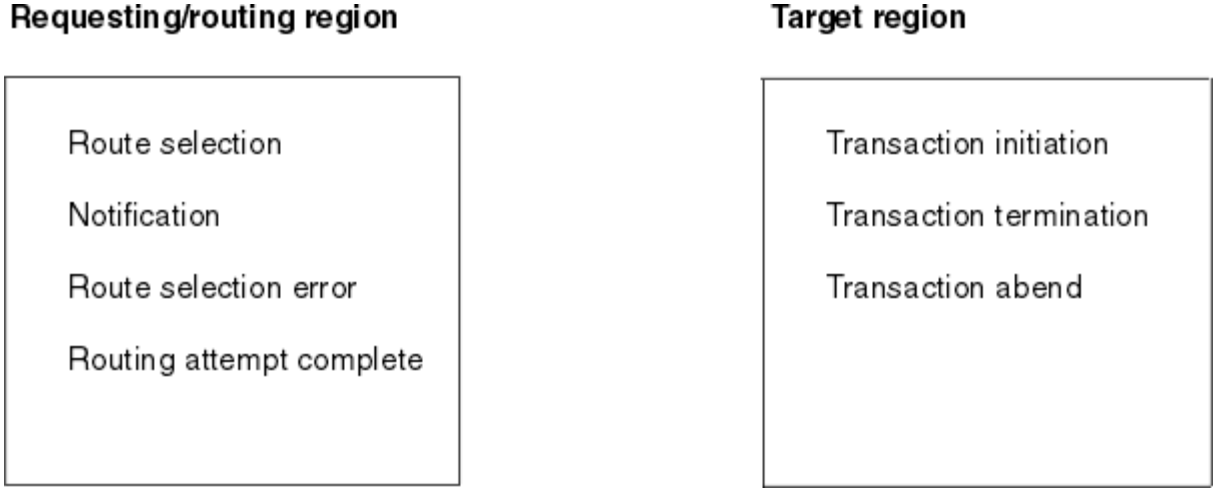


Figure 57. When and where the distributed routing program is invoked

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 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 228.

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.

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.
- 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.

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 routing program on the routing region has specified that it should be re-invoked 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 58 on page 230 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.

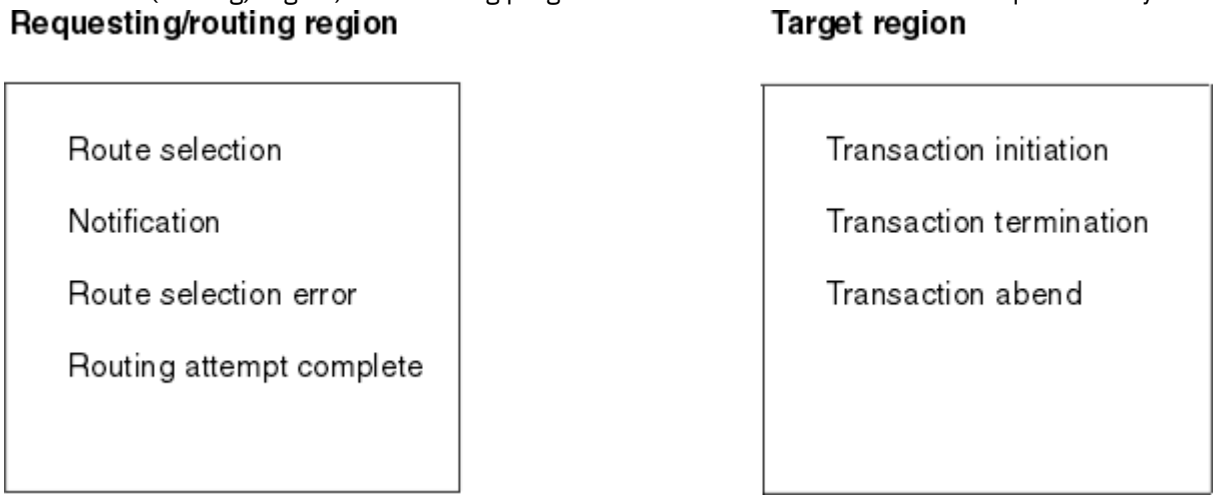


Figure 58. When and where the distributed routing program is invoked

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 231.

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” on page 231.

For information about writing an XICERES global user exit program, see Interval control EXEC interface program exits (XICEREQ, XICERES, and XICEREQC).

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.

Dynamic routing of inbound requests in a terminal handler

When the terminal handler of a service provider pipeline is one of the CICS-supplied SOAP message handlers, the target application handler program specified in container **DFHWS-APPHANDLER** is, in some cases, eligible for dynamic routing. All pipeline processing before the application handler program is always performed locally in the CICS region that received the SOAP message.

The transaction that runs the target application handler program is eligible for routing when one of the following conditions is true:

- The transaction under which the pipeline is processing the message is defined as DYNAMIC or REMOTE. This transaction is defined in the URIMAP that is used to map the URI from the inbound SOAP message.
- A program in the pipeline has changed the contents of container **DFHWS-USERID** from its initial value.
- A program in the pipeline has changed the contents of container **DFHWS-TRANID** from its initial value.
- A WS-AT SOAP header exists in the inbound SOAP message.

In all the preceding scenarios, a task switch occurs during the pipeline processing. The second task runs under the transaction specified in the **DFHWS-TRANID** container. This task switch provides an opportunity for dynamic routing to take place, but only if MRO is used to connect the CICS regions together. In addition, the CICS region that you are routing to must support channels and containers.

The routing only takes place if the TRANSACTION definition for the transaction named in **DFHWS-TRANID** specifies one of the following sets of attributes:

DYNAMIC(YES)
The transaction is routed using the distributed routing model, in which the routing program is specified in the **DSRTPGM** system initialization parameter.

DYNAMIC(NO) REMOTESYSTEM(*sysid*)
The transaction is routed to the system identified by *sysid*.

For more information about the routing of web service requests, see technote: Routing of provider mode CICS web services .

For applications deployed with the CICS web services assistant, there is a second opportunity to dynamically route the request, at the point where CICS links to the users program. The request is then routed using the dynamic routing model, in which the routing program is specified in the **DTRPGM** system initialization parameter. Eligibility for routing is determined, in this case, by the characteristics of the program. If you are using a channel and containers when linking to the program, you can only dynamically route the request to CICS regions that are at V3.1 or higher. If you are using a COMMAREA, this restriction does not apply.

When a 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 can, however, be statically routed from the target region to a third region.

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. For routing the request.
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 59 on page 233 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.

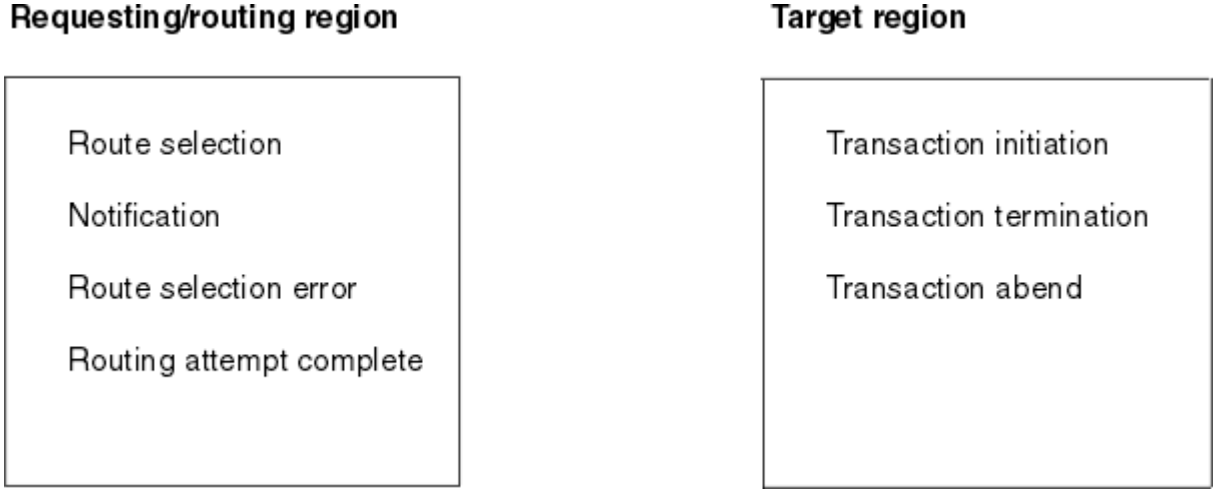


Figure 59. When and where the distributed routing program is invoked

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 234.

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.

Prerequisite: The distributed routing program must be specified in the **DSRTPGM** system initialization parameter in all potential target regions.

How can you re-invoke the routing program 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.

When the routing program is re-invoked on the target region?

If the routing program sets DYROPTER to 'Y', it is re-invoked on the target region under any of these situations:

- 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

What happens when the routing program is invoked on the target region?

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 236.

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.

- 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 211.

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 235.

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 as a result of the non-availability of connections to resource managers Db2, IMS, IBM MQ, or VSAM RLS. For more information, see Avoiding the storm drain effect.

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.

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.

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'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 228. 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.

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 Process and activity user IDsin the IBM Knowlege Center.
- 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 5 , 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:

a) Use the CICS Explorer **Regions** operations view.

b) Use the DSRTPGM system initialization parameter.

c) Use the **EXEC CICS SET SYSTEM DSRTPROGRAM** command. For programming information about this command, see SET SYSTEM.

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 CICSTS55.CICS libraries in which they can be found, are summarized in the following tables.

Table 19. Distributed routing programs

Language	Member name	Library
Assembler	DFHDSRP	SDFHSAMP
COBOL	DFHDSRP	SDFHCOB
PL/I	DFHDSRP	SDFHPL1
C	DFHDSRP	SDFHC370

Table 20. 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.

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 60 on page 245.

```
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 60. 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 CICSTS55.CICS.SDFHSAMP library. A corresponding copy book, DFHDBUCA, that maps the communication area, is in CICSTS55.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 "Assembling and link-editing user-replaceable programs" on page 330.

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 IBM MQ commands in a bridge exit so that CICS can get and put messages on IBM MQ queues.

For a detailed description of the 3270 bridge exit and its interfaces, see Introduction to the 3270 bridge.

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. The module source provides an example of how to set these options; look for the section that is commented as follows:

```
* WHERE DO THE PIPI RUNTIME OPTIONS COME FROM ?
    **
    **  1. CEEDOPT  - Installation default options
    **  2. DFHAPXPO - These runtime options
```

```
**  3. CICS - Xplink options forced by DFHAPLX
**  The options string passed to PIPI is shown at AP trace level 2.    *
```

DFHAPLX is an internal CICS module.

Analyzer programs

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Analyzer programs are associated with TCPIPSERVICE definitions. Their primary role is to interpret an HTTP request if a URIMAP definition specifies the use of an analyzer program, or if no URIMAP definition is present.

Analyzer programs cannot be invoked when CICS is an HTTP client, or for web service processing; they can only be invoked when CICS is an HTTP server. The role of analyzer programs in the CICS web support process for CICS as an HTTP server is described in HTTP request and response processing for CICS as an HTTP server. Enabling CICS web support for CICS as a HTTP server has information to help you plan your architecture for CICS as an HTTP server.

Relationship between analyzer programs and URIMAP definitions

Before CICS Transaction Server for z/OS, Version 3 Release 1, all HTTP requests for CICS as an HTTP server were interpreted by an analyzer program. Now, URIMAP resources are the strategic method for controlling the processing of HTTP requests. They replace key functions of the analyzer program in matching the URLs of requests to the application program that processes them, and specifying the use of a converter program and an alias transaction.

URIMAP definitions may, however, invoke an analyzer program for selected HTTP requests, to take over some of the processing stages, and to perform other actions such as monitoring or audit actions. The attributes of the URIMAP definition that reproduce analyzer functions, namely CONVERTER (converter program name), TRANSACTION (alias transaction), USERID (user ID for alias transaction), and PROGRAM (name of application program to process request), can be passed to the analyzer program, and the analyzer program can choose to override these.

You can choose to pass an HTTP request directly to an analyzer program without using a URIMAP definition, following the same process that CICS web support used before CICS Transaction Server for z/OS, Version 3 Release 1. However, without URIMAP definitions, if you want to change the way in which CICS responds to a particular HTTP request, you need to change the logic in the analyzer program. With URIMAP definitions, you can perform these changes dynamically as a system management task. Also note that if you continue to use an analyzer program instead of a URIMAP definition to handle requests, and you need to be compliant with HTTP/1.1 in this respect, you must code your analyzer program to perform URL comparison according to the rules stated in the HTTP/1.1 specification (RFC 2616).

Note: As supplied, the CICS-supplied sample analyzer program DFHWBADX and the CICS-supplied default analyzer program DFHWBAAX do not perform any analysis of a request when a matching URIMAP definition has been found for the request, even if the URIMAP specifies ANALYZER(YES).

If you select the analyzer program, the listener task cannot directly attach user transactions for fast arriving HTTP requests. For more information, see Processing HTTP requests by using directly attached user transactions.

Use of analyzer programs for error handling

Although an analyzer program is not now required in the processing path for every HTTP request, an analyzer program must still be specified for each TCPIPSERVICE resource that is used for CICS web support.

The name of the analyzer program is specified in the URM attribute of the resource definition. You can specify a different analyzer in each TCPIPSERVICE definition, or you can specify the same analyzer in more than one TCPIPSERVICE definition. If you are invoking an analyzer program from a URIMAP

definition, you cannot choose between different analyzer programs; you can only select whether or not to use the analyzer program specified for the TCPIPSERVICE definition.

The analyzer program specified for a TCPIPSERVICE definition is invoked to handle an HTTP request if CICS does not find a matching URIMAP definition for the request. This could be caused by a user error in typing a request URL, or because the appropriate URIMAP definition is not installed. (If the URIMAP definition exists but is disabled, the request is handled by a web error program, not the analyzer program.)

Because of this, as a minimum, the analyzer program specified for each TCPIPSERVICE definition should include a procedure to handle any HTTP request that it does not recognize, and provide a suitable error response. You may also identify specific requests that should have been handled by a URIMAP definition, and provide a more relevant error response. The output from an analyzer program in an error situation is passed to a web error program, which you can use to modify the HTTP response. Web error programs explains how to tailor these.

The CICS-supplied default analyzer program DFHWBAAX is the default when a TCPIPSERVICE definition specifies PROTOCOL(HTTP). DFHWBAAX provides basic error handling when all requests on the port should be handled by URIMAP definitions. It does not provide support for requests using the URL format that CICS web support used before CICS TS 3.1. If you need to provide handling in your analyzer program for requests that are not handled by URIMAP definitions, the analyzer program specified on your TCPIPSERVICE definition should be the CICS-supplied sample analyzer program DFHWBADX or your own customized analyzer program.

Use of analyzer programs for some non-web-aware applications, and for non-HTTP messages

Non-web-aware applications might function correctly when they are invoked directly from a URIMAP definition. However, some might be dependent on facilities that can only be provided for them by an analyzer program. The use of an analyzer program in the processing path for an HTTP request might be needed in the following circumstances:

- You are producing a response using a non-web-aware application and a converter program, and it needs to be flagged for pre-CICS TS Version 3 compatibility processing, because a web client requires a response identical with the response it would have received before CICS TS Version 3. (For example, user-written clients could experience problems with new error responses or additional HTTP headers.) This flag only works if the converter program produces the response manually in a block of storage. If the converter program uses the **EXEC CICS** WEB API commands to send the response, the flag has no effect.
- You are producing a response using a non-web-aware application and a converter program, and either the copy of the web client's request which is passed to the converter program in a block of storage, or an HTTP response which the converter program produces manually in a block of storage, requires nonstandard code page conversion. A converter program is not able to specify code page conversion settings for HTTP requests or responses that are passed in a block of storage. The standard settings used by CICS for code page conversion if no analyzer program is present in the processing path are described in “Writing a converter program” on page 257. If these standard settings are not suitable, or if code page conversion is not wanted, you can use an analyzer program in the processing path to specify alternative code page conversion settings. As an alternative to using an analyzer program, you could use the **EXEC CICS** WEB API commands in the converter program to examine the web client's request or to produce the response, instead of using the block of storage. In this case, code page conversion can be specified as usual on the **EXEC CICS** WEB API commands.

If you require an analyzer program to handle one of these situations, a URIMAP definition may be set up for the request, but it must specify the analyzer program.

For non-HTTP requests, which use the user-defined (USER) protocol on the TCPIPSERVICE definition, an analyzer program is always required to process the requests, and URIMAP definitions cannot be used. Web error programs explains how non-HTTP requests are processed.

Use of analyzer programs for additional processing

In situations where the use of an analyzer program in the processing path is optional, you might choose to use an analyzer program for reasons such as the following:

- You want to make dynamic changes to elements of the processing path, based on the content of the request. Each URL for a HTTP request is matched by a single URIMAP definition, which defines a single processing path. An analyzer program can interpret the content of the request and change elements such as the application program that handles the request, the involvement of a converter program, or the alias transaction and user ID used for the request.
- You want to introduce monitoring or audit actions into the process. An analyzer program is an appropriate location in which to do this.
- You are upgrading an existing CICS web support architecture from CICS TS Version 2, and your existing analyzer program provides additional functions that you want to maintain during request processing, such as passing information to a converter program.

"Writing an analyzer program" on page 249 explains the full range of functions that an analyzer program can perform.

Replacing analyzer programs with URIMAP definitions

You can replace the request processing functions of your analyzer program with URIMAP resource definitions, which can be changed and controlled by using CICS system programming commands.

URIMAP definitions can be used to match the URLs of requests and map them to application programs, and specify a converter program, alias transaction, and user ID. If your analyzer program provides extra functions, you can continue to use it instead of a URIMAP definition, or you can combine it with a URIMAP definition.

While you are migrating to the use of URIMAPs:

- You can introduce URIMAP resource definitions progressively for a few requests at a time. Depending on the type of processing carried out by your analyzer program, and the type of application that handles the request, you can choose whether to continue by using the analyzer program in the processing path for each request.
- You might prefer to select and publish new URLs for requests that are handled by URIMAP resource definitions, rather than retaining your existing URLs. When you are ready to discontinue the use of the old processing path for a request, you can set up a URIMAP definition to permanently redirect requests from the old URL to the new URL.
- Ensure that your analyzer program still contains basic handling procedures for unrecognized requests, even if it is no longer involved in the processing path for any requests. The analyzer program is still required on the TCPIPSERVICE definition, and receives requests in situations such as the user incorrectly typing a URL.

Using URIMAP definitions (without an analyzer program) qualifies requests for the direct attach route where CWBA or its transaction is attached directly by the socket listener task to process this request. This option reduces the CPU time that is required to process requests. For more information, see Processing HTTP requests by using directly attached user transactions.

Writing an analyzer program

You can write an analyzer program in Assembler, C, COBOL, or PL/I.

About this task

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Input and output parameters for an analyzer program are passed in a COMMAREA. Language-dependent header files, include files, and copy books which map the COMMAREA are described in Reference information for analyzer programs.

The full range of functions which an analyzer program can perform is as follows:

- Determine whether processing should continue for the request, or whether CICS should return an error response to the web client.
- Analyze the content of the request, and any parameters that have been passed to the converter program from a URIMAP definition, to determine which of the subsequent processing stages are required, and which CICS resources are needed to carry out each stage. (The **EXEC CICS** WEB API commands may be used during this analysis.)
- Specify the name of a converter program to process the request before it is passed to an application program. Converter programs are normally used with application programs that are not web-aware. A user token is provided for the analyzer program to communicate with the converter program, if required. The web client's request is passed to the converter program in a 32K block of storage indicated by a pointer in the parameter list. Converter programs explains the functions of a converter program.
- Specify the name of the user-written application program that is to process the request and provide the response.
- Specify the transaction ID of the alias transaction that handles the remaining stages of processing.
- Specify a user ID that is to be associated with the alias transaction.
- Specify or suppress code page conversion for the request passed to the converter program in the block of storage, and any response that the converter program constructs manually in a block of storage. This does not affect converter programs or user-written applications which use the **EXEC CICS** WEB API commands to view the HTTP request and produce the response; they request code page conversion directly from CICS. Code page conversion for CICS Web support explains the code page conversion process.
- Specify the flag, provided for upgrade purposes, that indicates where a non-web-aware application requires pre-CICS TS Version 3 compatibility processing. This does not affect converter programs or user-written applications which use the **EXEC CICS** WEB API commands to view the HTTP request and produce the response.
- Modify the request body. Any changes made are visible in the data passed to the converter program in the block of storage, but **not** to the **EXEC CICS** WEB API commands.

CICS supplies the default analyzer program DFHWBAAX, which is described in “CICS-supplied default analyzer program DFHWBAAX” on page 255, and the sample analyzer program DFHWBADX, which is described in “CICS-supplied sample analyzer program DFHWBADX” on page 255. If these analyzers do not meet your requirements, you need to write your own. You might be able to use DFHWBADX as an example.

All the user-replaceable programs must be local to the system in which CICS web support is operating. If you do not use autoinstall for programs, you must define and install program definitions for all user-replaceable programs used by CICS web support, including the analyzer and converter programs. If you use autoinstall for programs, you must ensure that user-replaceable programs are installed with the correct attributes. Note that your analyzer programs must be defined with EXECKEY(CICS).

For more information about writing user-replaceable programs, see Customizing with user-replaceable programs in Developing system programs.

Input to an analyzer program

Input parameters are passed to the analyzer program in a COMMAREA, giving information about the nature and content of the request, and any input supplied by a URIMAP definition. The analyzer program can choose to accept these values and pass them on as output parameters, or it can dynamically override them based on its analysis of the content of the request.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Parameters for analyzer programs has a listing and technical descriptions of all the parameters in the COMMAREA.

The input parameters include the following items, or a pointer to them:

- An eye-catcher for an analyzer parameter list.
- The colon hexadecimal or dotted decimal IP address of the client and of the server (CICS as an HTTP server).
- An indicator of whether the request is an HTTP request.
- An indicator of whether a matching URIMAP definition was found for the request. If this indicator is positive, the URIMAP definition might have passed additional input parameters to the analyzer program.
- The HTTP version.
- The request method.
- The host name specified for the request, taken from the Host header or, for an absolute URI, from the request URL. For HTTP/1.1 requests, a host name is required, so this parameter is always passed to the analyzer. For HTTP/1.0 requests, a host name might not be supplied.
- The path component of the URL.
- Any query string that was specified for the request.
- The HTTP headers for the request.

  If the request has been sent using chunked transfer-coding, any trailing headers are not passed to the analyzer program with the main request headers.
- The request body, or as much of the request body as fits into a 32 KB block of storage. This request body is a pointer to the separate block of storage containing the request.

For HTTP requests received on a connection using SSL client authentication, the following parameter is also passed:

- The user ID obtained from the client certificate.

If a matching URIMAP definition is found for the request and it invoked the analyzer program, the following parameters from the URIMAP definition are passed to the analyzer program, if they are present in the URIMAP definition:

- The name of the recommended converter program to process the request before it is passed to an application program (CONVERTER attribute in the URIMAP definition).
- The name of the recommended user-written application program to process the request and provide the response (PROGRAM attribute in the URIMAP definition).
- The transaction ID of the recommended alias transaction to cover the remaining stages of processing (TRANSACTION attribute in the URIMAP definition).
- The recommended user ID that is to be associated with the alias transaction (USERID attribute in the URIMAP definition). This user ID can be overridden if a user ID is supplied by the client.

The **wbra_urimap** input parameter can be used to test whether or not a URIMAP definition was used in the processing path for the request.

If you are using an analyzer program instead of a URIMAP definition to handle requests, and you need to comply with HTTP/1.1 in this respect, you must code your analyzer program to perform URL comparison according to the rules stated in the HTTP/1.1 specification. Under these rules, scheme names and host names are compared case-insensitively, but paths are compared case-sensitively. All components are unescaped before comparison. When CICS compares URLs to URIMAP definitions, it follows these rules.

You can also use the **EXEC CICS** WEB API commands to examine the HTTP request, if preferred. Using the **EXEC CICS** WEB commands can increase the accuracy and completeness of your analysis of the request, particularly when examining the HTTP headers, which are subject to wide variation in content and usage. The **EXEC CICS** WEB commands also simplify the process of locating and extracting query string or formfield information from a request, which can be a determine the subsequent processing.

You can use the EXTRACT TCPIP command to obtain the following information about the client request that is being processed:

- The IP address of the web client
- The host name of the web client, as known by the DNS server
- The number of the port on which the web client sent its connection request
- The IP address of the server; that is, CICS as an HTTP server
- The type of authentication in use
- The level of SSL support in use
- The TCPIPSERVICE resource definition associated with the request

Output from an analyzer program

An analyzer program provides output in a COMMAREA. The output includes a response code, and a range of optional output parameters that can be used to specify further processing stages and to share information with a converter program.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Parameters for analyzer programs has a listing and technical descriptions of all the parameters in the COMMAREA.

The analyzer program must provide the following output in its COMMAREA:

- A response code.
  - If your analyzer program returns a response code of URP_OK, processing continues with the next step.
  - If your analyzer program returns any other value, CICS returns an error response to the web client. The response can be modified with a user-replaceable web error program. CICS web support default status codes and error responses tells you how the return codes from the analyzer map to the status codes that CICS returns to the web client.

The analyzer program may also provide the following outputs:

- The name of a converter program that is to be used to process the request before it is passed to a user-written application program.
  - If a converter program name was input from a URIMAP definition, you can accept or override this.
  - If the analyzer indicates that a converter program is not required, the first 32K bytes of the request is passed to the user-written application program in a block of storage. A web-aware application can ignore this and use the **EXEC CICS** WEB API commands to read the request.
- The name of an application program that is to process the request and provide the response.
  - If a program name was input from a URIMAP definition, you can accept or override this.
  - If you are using a converter program, the converter program can specify or override the program name. A converter can be used in this way to involve more than one program in processing the request.
- The transaction ID of the alias transaction that is to cover the remaining stages of processing. If a transaction ID was input from a URIMAP definition, you can accept or override this.
- The user ID that is to be associated with the alias transaction. If a user ID was input from a URIMAP definition, you can accept or override this. This is how CICS determines the user ID if you do not specify one:
  - If a user ID was input from a URIMAP definition, that is used.
  - If the HTTP request uses SSL with client authentication, the user ID is obtained from the client certificate.
  - In other cases, the CICS default user ID is used.

- Parameters relating to code page conversion of the 32K block of storage containing the request, and to code page conversion of the response body, if the converter program produces it manually in a block of storage.

  Note: This does not affect converter programs or user-written applications which use the **EXEC CICS** WEB API commands to view the HTTP request and produce the response; they request code page conversion directly from CICS.

  You can specify the parameters for conversion of the block of storage containing the request in one of two ways:

  – As a pair of parameters specifying the character set used by the web client (`wbra_characterset`), and the host code page suitable for the application program (`wbra_hostcodepage`). Specifying the parameters in this way means that an entry in the code page conversion table (DFHCNV) is not required.

  – As a key for an entry in the DFHCNV code page conversion table (`wbra_dfhcnv_key`). This is not recommended, except for upgrade purposes.

  If you do not specify any of these parameters, the default behavior is for CICS to convert a text message using the standard settings described in "Analyzer programs" on page 247. If you want to suppress code page conversion for the request and response in the block of storage, set `wbra_dfhcnv_key` to nulls or blanks.
- The flag that indicates where a non-web-aware application that uses a converter program requires pre-CICS TS Version 3 compatibility processing (wbra_commarea). This flag is provided for upgrade purposes. It can be used only by applications that do not use the **EXEC CICS** WEB API commands (that is, they produce the response manually in a block of storage), in the specific circumstance where the web client needs a response that is identical with the response it would have received before CICS TS Version 3. Setting this flag means that:

  – CICS does not add any of the response headers that are normally inserted for HTTP/1.1 messages. Only the headers that were sent to clients before CICS Transaction Server for z/OS, Version 3 Release 1 are used.

  – If error processing is required, CICS sends an error response that is suitable for, and labeled as, an HTTP/1.0 response, regardless of the HTTP version of the web client. CICS would normally reply to a HTTP/1.1 client with an HTTP/1.1 error response, but this might mislead the client into thinking that the application would normally send an HTTP/1.1 response.
- An eight byte user token, used to share information between the analyzer and converter programs. "Sharing data between analyzer and converter programs" on page 253 explains how this works.
- A modified value for the request body length.

The analyzer can modify the contents of the request:

- The modified data can be shorter than, or of the same length as the original data. The request body cannot be lengthened.
- Any changes made are visible in the data passed to the converter program, but **not** to the **EXEC CICS** WEB API commands.

Sharing data between analyzer and converter programs

CICS passes three parameters between the analyzer and the converter programs that enable data to be shared by these processing stages.

The user_data pointer
This parameter contains the address of a 32K block of storage that is passed from stage to stage. On entry to the analyzer program, the pointer points to a block of storage containing the HTTP request. On completion of the encode function of the converter program, CICS web support uses it to locate the block of storage containing the HTTP response, unless the **EXEC CICS** WEB API commands have been used to produce a response instead.

You must not change the value of the pointer in the analyzer program, although you can modify the contents of the block of storage addressed by the pointer.

Between the converter program and the user-written application program, you can pass the pointer unchanged from one stage to another, or you can issue a GETMAIN command in one program and pass the address of the newly acquired storage in the pointer.

The user_data length
This parameter is the length of the block of storage addressed by the user_data pointer.

The user token
The user token is an 8–byte field which is shared by the analyzer program and the converter program. It can contain any information you want:

- You can pass small quantities of shared information directly in the user token.
- To pass larger quantities, you can issue a GETMAIN command in one program, to acquire storage for a shared work area. Use the user token to pass the address of the shared storage.

You can change the contents of the user token in each program: for example, the user token can have one meaning when passed from the analyzer program to the decode function of the converter program, and a different meaning when passed to the encode function.

The analyzer program can modify any of the parameters in the parameter list which is passed to the converter program. The pointers cannot be changed, but the data indicated by the pointer can be changed. The length of each field must not change.

Note: The analyzer and converter programs execute under different CICS tasks. Therefore, if you issue a GETMAIN command in the analyzer program, you must code the SHARED option if the storage is to be visible in the converter program. In general, storage acquired with the SHARED option is not freed automatically by CICS, so you must issue a FREEMAIN command when your programs no longer need the storage. However, CICS will free the storage addressed by the user_data pointer after the HTTP response has been sent to the web client.

Selecting escaped or unescaped data from an analyzer program

The HTTP request which is passed to the analyzer program for parsing is in its escaped form. Reserved or excluded characters in the URL, or in form data in the message body, are presented as a %xx sequence, wherexx is the ASCII hexadecimal representation of the reserved character. The analyzer can pass the request in a 32K block of storage to subsequent processing stages in its escaped form, with the escape sequences still present, or in its unescaped form, with the escape sequences converted back to the original characters. Web-aware application programs using the **EXEC CICS** WEB API commands do not use this mechanism to receive the response, and they request unescaping directly from CICS.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Defining local resources for DPL explains escaping and its purpose. Escaping and unescaping only applies to the following elements of the HTTP request:

- The URL portion of the request line, including any query string. The query string might be data from a form with the GET method.
- Form data returned from a form with the POST method and the default encoding **application/x-www-form-urlencoded**. This data is presented in the message body. Choosing the access method for MRO explains more about form data.

If the request in the 32K block of storage is to be passed on in unescaped form, the analyzer can convert the data from escaped to unescaped form, or have CICS perform the conversion.

- To pass the request in escaped form, set WBRA_UNESCAPE to WBRA_UNESCAPE_NOT_REQUIRED in your analyzer. WBRA_UNESCAPE_NOT_REQUIRED is the default value.
- To pass the request in unescaped form and have CICS perform the conversion, set WBRA_UNESCAPE to WBRA_UNESCAPE_REQUIRED in your analyzer.

- To pass the request in unescaped form after the analyzer has performed the conversion, set WBRA_UNESCAPE to WBRA_UNESCAPE_NOT_REQUIRED.

Web-aware application programs using the **EXEC CICS** WEB API commands do not use the COMMAREA mechanism to receive and send the response, and they request unescaping directly from CICS. For web-aware applications that use the **EXEC CICS** WEB API commands, when you extract form data from a request using the WEB READ FORMFIELD command or form field browsing commands, CICS performs the unescaping, and the data is returned in its unescaped form. When you extract a query string from a request using the WEB EXTRACT command, the data is returned in its escaped form.

If you are writing an application with a COMMAREA interface that can be run either through CICS web support or through the CICS business logic interface, ensure that WBRA_UNESCAPE is set to WBRA_UNESCAPE_NOT_REQUIRED, and that any unescaping is delegated to the application. If this is not done, the application is passed unescaped data by the CICS business logic interface, and escaped data by CICS web support, which might cause unpredictable results.

CICS-supplied default analyzer program DFHWBAAX

CICS supplies a default analyzer program, DFHWBAAX. DFHWBAAX provides an error handling function for TCPIPSERVICE resource definitions that are used for CICS web support. It is suitable for use when all of the requests using a port are handled using URIMAP definitions.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

CICS supplies the source code for DFHWBAAX in Assembler only.

DFHWBAAX is the default analyzer program for a TCPIPSERVICE definition that specifies PROTOCOL(HTTP).

DFHWBAAX receives the same input and output parameters as a standard analyzer program, in a COMMAREA. As supplied, it does not make use of most of these parameters, and it does not provide support for requests using the URL format that CICS web support used before CICS TS 3.1. Instead, it takes simplified action as follows:

- DFHWBAAX does not carry out further processing when a matching URIMAP definition has been found for the request, even if the URIMAP specifies ANALYZER(YES). It uses the `wbra_urimap` input parameter to test for the presence of a URIMAP definition, and if the result is positive, returns without performing any analysis on the request URL. This means that the settings specified in the URIMAP definition for the alias transaction, converter program (if used), and application program are automatically accepted and used to determine subsequent processing stages.
- If no matching URIMAP definition is found, DFHWBAAX gives control to the user-replaceable web error transaction program DFHWBERX to produce an error response. This is achieved by setting DFHWBERX as the application program to handle the request, using the `wbra_server_program` output parameter. DFHWBAAX does not make any other changes to the COMMAREA. On receiving control, DFHWBERX provides either an HTTP response with a 404 (Not Found) status code, or a SOAP fault response, depending on the request made by the web client.

DFHWBAAX uses a standard range of responses, **URP_OK**, **URP_EXCEPTION**, and **URP_INVALID**. No reason values are architected for DFHWBAAX as supplied. Note that if the response is other than **URP_OK**, this indicates an error in processing, and control is passed to the user-replaceable web error program DFHWBEP, rather than the web error application program DFHWBERX.

CICS-supplied sample analyzer program DFHWBADX

CICS supplies a working sample analyzer program, DFHWBADX. If you need to provide request handling through your analyzer program, as well as or instead of through URIMAP definitions, you can use DFHWBADX as a starting point for writing your own analyzer program.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

CICS supplies the source code in several languages:

- DFHWBADX (Assembler)
- DFHWBAHX (C)
- DFHWBALX (PL/I)
- DFHWBAOX (COBOL)

As supplied, DFHWBADX does not perform any analysis of a request when a matching URIMAP definition has been found for the request, even if the URIMAP specifies ANALYZER(YES). This means that the settings specified in the URIMAP definition for the alias transaction, converter program and application program are automatically accepted and used to determine subsequent processing stages.

DFHWBADX uses the `wbra_urimap` input parameter to test for the presence of a URIMAP definition, and if the result is positive, returns without performing any analysis on the request URL. If you write your own analyzer program and want it to interact with a URIMAP definition, do not copy this aspect of DFHWBADX's processing. You may want to test the `wbra_urimap` input parameter in order to modify your analyzer program's processing in other ways. For example, you could test the parameter to decide whether to perform analysis based on the input parameters from the URIMAP definition, or to perform analysis directly on the request URL.

How DFHWBADX interprets a request URL

DFHWBADX interprets HTTP requests in which the path component of the URL has the following syntax:

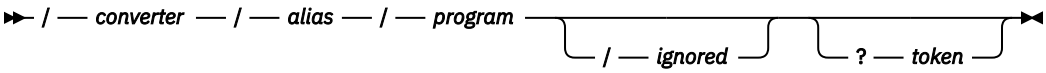


Figure 61. Syntax of path component interpreted by DFHWBADX

All fields processed by the analyzer program are translated to uppercase. After translation:

converter
Specifies the name of the converter program to be used for the request. It can be up to eight characters in length.

As a special case, the four character value `'CICS'` denotes that no converter program is used. See Converter programs for information on how to use converter programs with URIMAP definitions.

alias
Specifies the transaction ID of the alias transaction for subsequent request processing. It can be up to four characters in length.

program
Specifies the name of the CICS application program that is to be used to service the request. It can be up to eight characters in length.

ignored
This part of the path is ignored by DFHWBADX (but may be used by the converter program or the application program).

token
The initial eight bytes specify the user token that is passed to the converter program. Data following the first eight bytes of the token is ignored by DFHWBADX (but may be used by the converter program or the application program).

In the example path `/cics/cwba/dfh$wb1a`:

- No converter program is used.
- The alias transaction is CWBA.
- The CICS application program is DFH$WB1A.

In addition to the outputs derived from the original HTTP request, DFHWBADX sets the following outputs:

- The code page conversion template is DFHWBUD. This template is defined in sample conversion table DFHCNVW$, and converts data between the ASCII Latin-1 character set (code page ISO 8859–1) and the EBCDIC Latin character set (code page 037). The sample conversion table can be used without any configuration, but note that the output parameters `wbra_characterset` and `wbra_hostcodepage` can be used in place of the `wbra_dfhcnv_key` output parameter to provide greater control and avoid the use of a conversion table.
- DFHWBADX passes the request in escaped form, and sets WBRA_UNESCAPE_NOT_REQUIRED.

Responses from DFHWBADX

The meanings of the responses produced by DFHWBADX are as follows:

URP_OK
The analyzer found that the request conformed to the default HTTP request format, and generated the appropriate outputs for the alias.

URP_EXCEPTION
The analyzer found that the request did not conform to the default format. A reason code is supplied as follows:

1
The length of the resource was less than 6. (With the URL format recognized by DFHWBADX, the shortest possible resource specification is `/A/B/C`, asking for program C to be run under transaction B with converter A.) This response and reason are the ones used when the incoming request is not an HTTP request.

2
The resource specification did not begin with a "/".

3
The resource specification contained one "/", but fewer than three of them.

4
The length of the converter name in the resource specification was 0 or more than 8.

5
The length of the transaction name in the resource specification was 0 or more than 4.

6
The length of the CICS application program name in the resource specification was 0 or more than 8.

The response and reason codes are displayed in message DFHWB0723. An error response with a 400 (Bad Request) status code is returned to the web client. This can be modified with the user-replaceable web error program DFHWBEP.

URP_INVALID
The eye-catcher was invalid. This indicates an internal error.

Writing a converter program

To write a converter program, you need to construct decode and encode functions, and consider code page conversion.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

You can write a converter program in Assembler, C, COBOL, or PL/I. Language-dependent header files, include files, and copy books are described in Reference information for converter programs.

Decode function: viewing and processing the HTTP request

The decode function of the converter program receives the HTTP request from the web client, together with a parameter list giving more information about the request. The HTTP request is passed to the

converter program in a 32K block of storage, which is indicated by a pointer in the parameter list. The request has already been divided into separate elements, such as the method, request headers and body. (Note that if the request is too long to fit into the block of storage, the remainder of the data is not passed to the converter program.) If an analyzer program is used in the processing path, the analyzer program might have modified the content of the request.

In a converter program for CICS web support, you can use **EXEC CICS** WEB API commands to examine the HTTP request, if you prefer. The WEB EXTRACT command retrieves information about the request (such as the method and version). The WEB READ HTTPHEADER command or the HTTPHEADER browsing commands can be used to read the HTTP headers. The WEB RECEIVE command can be used to receive the body of the request. If you use any of the **EXEC CICS** WEB API commands, note that these commands return the original information from the web client's request, and you cannot use them to see any modifications that an analyzer program has made. Changes by an analyzer program are only visible in the parameter list and block of storage passed directly to the converter program.

The name of the user-written application program that should provide data for the response is supplied in the parameter list, either taken from the URIMAP definition for the request, or set by the analyzer program. If an analyzer program is used, it can provide additional information directly to the converter program in a user token.

Using the information which you have obtained about the web client's request, the decode function of the converter program needs to:

- Determine whether processing should continue for the request, or whether CICS should return an error response to the web client.
- Specify the name of the user-written application program that is to process the request and provide the response. If the name has already been input from a URIMAP definition or by an analyzer program, the converter program can accept or change this.
- Construct the COMMAREA that is passed to the user-written application program. The COMMAREA includes data from the web client's request which has been converted into an acceptable input format for the application program. The block of storage containing the HTTP request can be reused, or a new COMMAREA can be specified.

Encode function: producing the response

When the user-written application program has carried out its processing using the input supplied by the converter program, the encode function of the converter program receives an output COMMAREA from the application program. Using this data, the encode function of the converter program needs to:

- Invoke further application programs, if more than one application program is needed to supply data. To do this, the encode function sets the loop response to call the decode function again. The decode function changes the name of the application program, and supplies appropriate input in a COMMAREA. The output is returned to the encode function again. “Calling more than one application program from a converter program” on page 261 has more information about this.
- Construct an HTTP response to be sent to the web client.

In a converter program for CICS web support, you can use **EXEC CICS** WEB API commands to produce and send the response to the web client. The WEB WRITE HTTPHEADER command can be used to write HTTP headers for the response. The WEB SEND command can be used to assemble and send the response.

Alternatively, the converter program can construct the HTTP response manually in a buffer of storage, and return this to CICS for sending to the web client. The response must contain an HTTP version, status code, status text, any HTTP headers that are required, and the message body. The format of the response should be compliant with the HTTP protocol specification to which you are working (HTTP/1.0 or HTTP/1.1). To obtain a buffer of storage for the HTTP response, you can:

- Issue a GETMAIN command to obtain storage.
- Use storage acquired in an earlier stage of processing (such as the analyzer program).
- Construct the response in the COMMAREA returned by the user-written application program.

The first word of the area used for the response must contain the length of the area (that is, the length of the HTTP response plus 4). On exit from the encode function of the converter program, the data pointer in the parameter list must point to this block of storage. (If you use **EXEC CICS** WEB API commands to send the response instead, CICS ignores and discards any block of storage indicated by this pointer.)

Whichever method you use to construct the HTTP response, CICS normally inserts some HTTP headers suitable for an HTTP/1.0 or HTTP/1.1 response, which are listed in HTTP header reference for CICS web support. If the response produced by the converter program already contains these headers, CICS does not replace them. If the response has been flagged by an analyzer program for pre-CICS TS Version 3 compatibility processing, because a web client requires a response identical with the response it would have received before CICS TS Version 3, only the headers that were sent to clients before CICS Transaction Server for z/OS, Version 3 Release 1 are used. This flag only works if the converter program produces the response manually in a block of storage. If the converter program uses the **EXEC CICS** WEB API commands to send the response, the flag has no effect.

Code page conversion

When you use **EXEC CICS** WEB API commands in a converter program to view the HTTP request and produce the response, code page conversion takes place as you specify in the commands, in the same way as for any other program which uses the **EXEC CICS** WEB API commands.

A converter program is not able to specify code page conversion settings for the HTTP request passed to it in the 32K block of storage. If a converter program is invoked directly from a URIMAP definition, and the headers for the web client's request indicate that the message body is text, CICS converts the message body supplied in the block of storage using the following standard settings:

- For the character set, if the web client's request has a Content-Type header naming a character set supported by CICS, that character set is used. If the web client's request has no Content-Type header or the named character set is unsupported, the ISO-8859-1 character set is used.
- For the host code page, CICS uses the default code page for the local CICS region, as specified in the LOCALCCSID system initialization parameter.

If these standard settings are not suitable, or if code page conversion is not wanted, either use an analyzer program in the processing path to specify alternative code page conversion settings, or use the **EXEC CICS** WEB API commands to handle the request.

If your converter program constructs the HTTP response manually in a buffer of storage, CICS mirrors the code page conversion that was carried out for the request passed in the 32K block of storage. The response is sent to the web client using the character set and host code page settings specified by the analyzer program, or in the absence of an analyzer program, the standard settings described in this topic. If the analyzer program suppressed code page conversion for the request, no code page conversion is carried out for the response body. If this outcome is not suitable, use the **EXEC CICS** WEB API commands to produce the response instead.

Converter programs for the CICS business logic interface

When you use a converter program with the CICS business logic interface, there are restrictions which might affect how you construct the COMMAREA that is passed to the user-written application program, and the buffer of storage containing the response. For more information, see Offset mode and pointer mode.

Do not use **EXEC CICS** WEB API commands in a converter program which is written for the CICS business logic interface.

Input parameters for converter program decode function

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Input parameters are passed to the decode function in a parameter list.

The parameters include:

- The IP address of the web client.
- A pointer to the HTTP version of the web client's request.
- A pointer to the request method.
- A pointer to the path component of the URL.
- A pointer to the HTTP headers for the request.
- A pointer to the entity body of the request message.
- The name of the CICS application program that provides data for the request (as set by the analyzer program, or specified in the URIMAP definition).
- An eight byte user token, used to share information between the analyzer and converter programs. See “Sharing data between analyzer and converter programs” on page 253.
- An iteration counter which records the number of times the decode function has been entered for each HTTP request. The counter is set to 1 before the decode function is called for the first time, and is incremented before it is called on each subsequent occasion.
- An indication of whether the address of the entity body can be the target of a FREEMAIN command.

The analyzer program can change the values of any of these parameters before passing the parameter list to the converter program. If you want to examine the original request from the web client, use the **EXEC CICS** WEB API commands in the converter program.

Output parameters for converter program decode function

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The decode function must provide the following outputs in a COMMAREA: a response code, and the address and length of the COMMAREA.

- A response code (optionally qualified by a reason code).

  If the decode function returns a response code of URP_OK, processing continues with the next step.

  If the decode function returns any other value, the HTTP request is rejected with an error response. For details of the response made by CICS in this situation, see CICS web support default status codes and error responses.
- The address and length of the COMMAREA passed to the user-written application program. If no application program is called, the COMMAREA is passed unchanged to the encode function.

The decode function may also provide the following outputs:

- The name of the user-written application program that is to provide data for the request. If the analyzer program supplied a name, the converter program can change it, or specify that no application program should be called.
- An eight byte user token, used to share information between the analyzer and converter programs. See “Sharing data between analyzer and converter programs” on page 253.

Input parameters for converter program encode function

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Input parameters are passed to the encode function in a COMMAREA.

The parameters include:

- The address and length of the COMMAREA returned by the user-written application program. If no application program was called, the COMMAREA is passed unchanged from the decode function.

- An eight byte user token, used to share information between the analyzer and converter programs. See “Sharing data between analyzer and converter programs” on page 253.
- An iteration counter that records the number of times the encode function has been entered for each HTTP request. The counter is set to 1 before the encode function is called for the first time, and is incremented before it is called on each subsequent occasion.

Output parameters for converter program encode function

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The encode function can provide the following outputs: a response code, the address of the storage buffer, the length of the HTTP response and an eight byte user token.

- A response code (optionally qualified by a reason):
  - If the encode function returns a response code of URP_OK, CICS sends the supplied HTTP response to the web client, unless you have already used the **EXEC CICS** WEB API commands to do this.
  - If the encode function returns a response code of URP_OK_LOOP, processing continues with the decode function. See “Calling more than one application program from a converter program” on page 261 for more information.
  - If the encode function returns any other value, the HTTP request is rejected with an error response. For details of the response made by CICS in this situation, see CICS web support default status codes and error responses.
- If you have constructed the HTTP response manually in a buffer of storage, the address of the buffer of storage, and the length of the HTTP response. The first word of the buffer must contain the length of the data (that is, the length of the HTTP response plus 4). If you use **EXEC CICS** WEB API commands to send the response instead, CICS ignores and discards any block of storage indicated by this pointer.
- An eight byte user token, used to share information between the analyzer and converter programs. See “Sharing data between analyzer and converter programs” on page 253.

Calling more than one application program from a converter program

Sometimes, the data you need to construct the response to an HTTP request comes from more than one user-written application program.

About this task

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

When this is the case, you can repeat the following sequence as necessary:

- The converter's decode function.
- An application program.
- The converter's encode function.

Do this by setting the response to URP_OK_LOOP in the encode function. When the HTTP response is complete, set the response to URP_OK.

When the decode function is called on the second and subsequent occasions, the following input parameters are not available:

- The HTTP version.
- The method.
- The path component of the URL.
- The request headers.
- The entity body.

However, you can use the WEB EXTRACT command to retrieve the same information.

Use the data pointer in the parameter list and the user token to share data between the decode and encode functions. When the encode function is called for the last time, if you are constructing the HTTP response manually in a buffer of storage, make sure that the data pointer (**encode_data_ptr**) addresses a valid HTTP response. If you are using **EXEC CICS** WEB API commands to produce and send the response, do so at this stage; in this situation, CICS ignores and discards any block of storage indicated by this pointer.

Chapter 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 z/Architecture Principles of Operation.

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.

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 Reset characteristics of statistics counters.

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 Introduction to CICS statistics.

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.

Figure 62 on page 265 illustrates the components of a CICS statistics record. Each of these is described in the sections that follow.

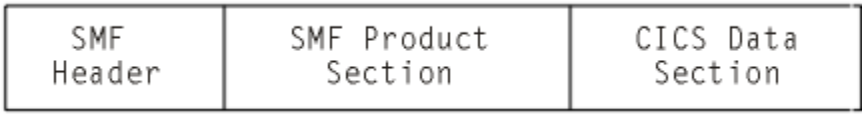


Figure 62. Format of an SMF type 110 statistics record

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 63 on page 266.

```
*        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 63. 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, see z/OS MVS System Management Facilities (SMF) .
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 statistics data section in a CICS statistics record consists of multiple statistics records.

The format of the CICS statistics data section is shown in Figure 64 on page 267.

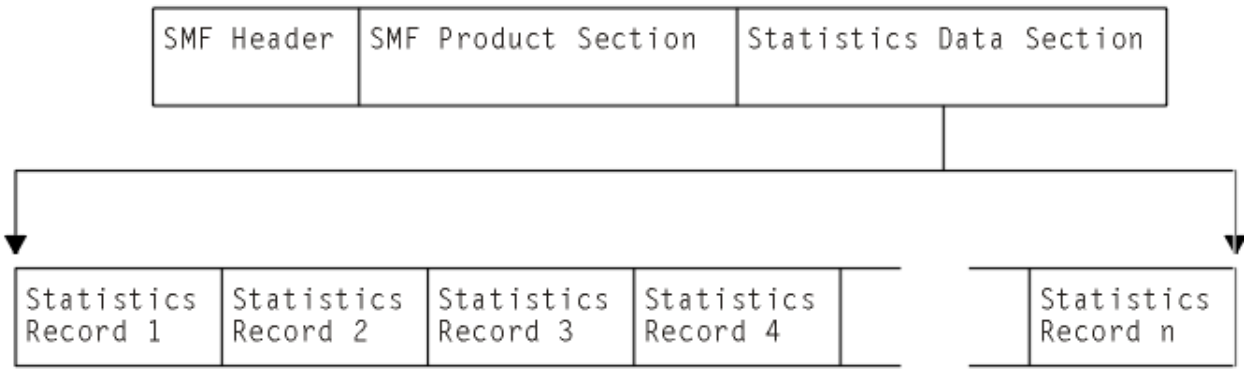


Figure 64. Format of the statistics data section

If the data records are incomplete, the flag field SMFSTICD is set to YES. In this 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 65 on page 267.

```
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 65. 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 Table 21 on page 267).

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 Table 21 on page 267. For further guidance information about the fields within the statistics data records, see CICS statistics in DSECTS and DFHSTUP report.

STIVERS
The version of the record. STIVERS takes the value 1 for this release of CICS.

Statistics data record copybooks related to STID name and value

Table 21. Statistics data record copybooks related to STID name and value

STID symbolic name	STID value	Copybook	Type of record
STIXMG	10	DFHXMGDS	Transaction manager (Globals) id

Table 21. Statistics data record copybooks related to STID name and value (continued)

STID symbolic name	STID value	Copybook	Type of record
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

Table 21. Statistics data record copybooks related to STID name and value (continued)

STID symbolic name	STID value	Copybook	Type of record
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
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
STIMPR	145	DFHMPRDS	POLICYs (Resource) id
STIPGP	146	DFHPGPDS	JVM programs - private

Table 21. Statistics data record copybooks related to STID name and value (continued)			
STID symbolic name	**STID value**	**Copybook**	**Type of record**
STIPGE	147	DFHPGEDS	Program definitions - private
STIMQR	148	DFHMQRDS	MQMONITORs (Resource) id
STIASG	149	DFHASGDS	ASYNCSERVICE (Global) id
STINDJ	150	DFHSJNDS	NODEJSAPP (Resource) id

TS data sharing statistics related to STID

The TS data sharing statistics use no symbolic names, but relate to the STID values as follows:

Table 22. TS data sharing statistics related to STID			
STID symbolic name	**STID value**	**Copybook**	**Type of record**
-	121	DFHXQS1D	TS server list structure stats id
-	122	DFHXQS2D	TS buffer stats id
-	123	DFHXQS3D	TS storage stats id

Coupling facility data table server statistics related to STID

The coupling facility data table server statistics use no symbolic names, but relate to the STID values as follows:

Table 23. Coupling facility data table server statistics related to STID			
STID symbolic name	**STID value**	**Copybook**	**Type of record**
-	126	DFHCFS6D	CFDT server list stats
-	127	DFHCFS7D	CFDT buffer stats id
-	128	DFHCFS8D	CFDT request stats id
-	129	DFHCFS9D	CFDT storage stats id

Named sequence server statistics related to STID

The named sequence number server statistics use no symbolic names, but relate to the STID values as follows:

Table 24. Named sequence server statistics related to STID			
STID symbolic name	**STID value**	**Copybook**	**Type of record**
-	124	DFHNCS4D	NC server list structure stats id
-	125	DFHNCS5D	NC server storage stats id

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 "Global user exit programs" on page 1.

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). For information about how to interpret the report produced by DFHSTUP, see CICS statistics in DSECTS and DFHSTUP report.

IBM Z® Decision Support
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 IBM Z Decision Support, see IBM Z Decision Support 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 "CICS statistics record format" on page 265.

Structure and content of CICS TS format journal records

Note:

For **SMF records**, see "Format of journal records written to SMF" on page 303.

This section does not apply to journal records written to an SMF data set.

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 66 on page 272 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.

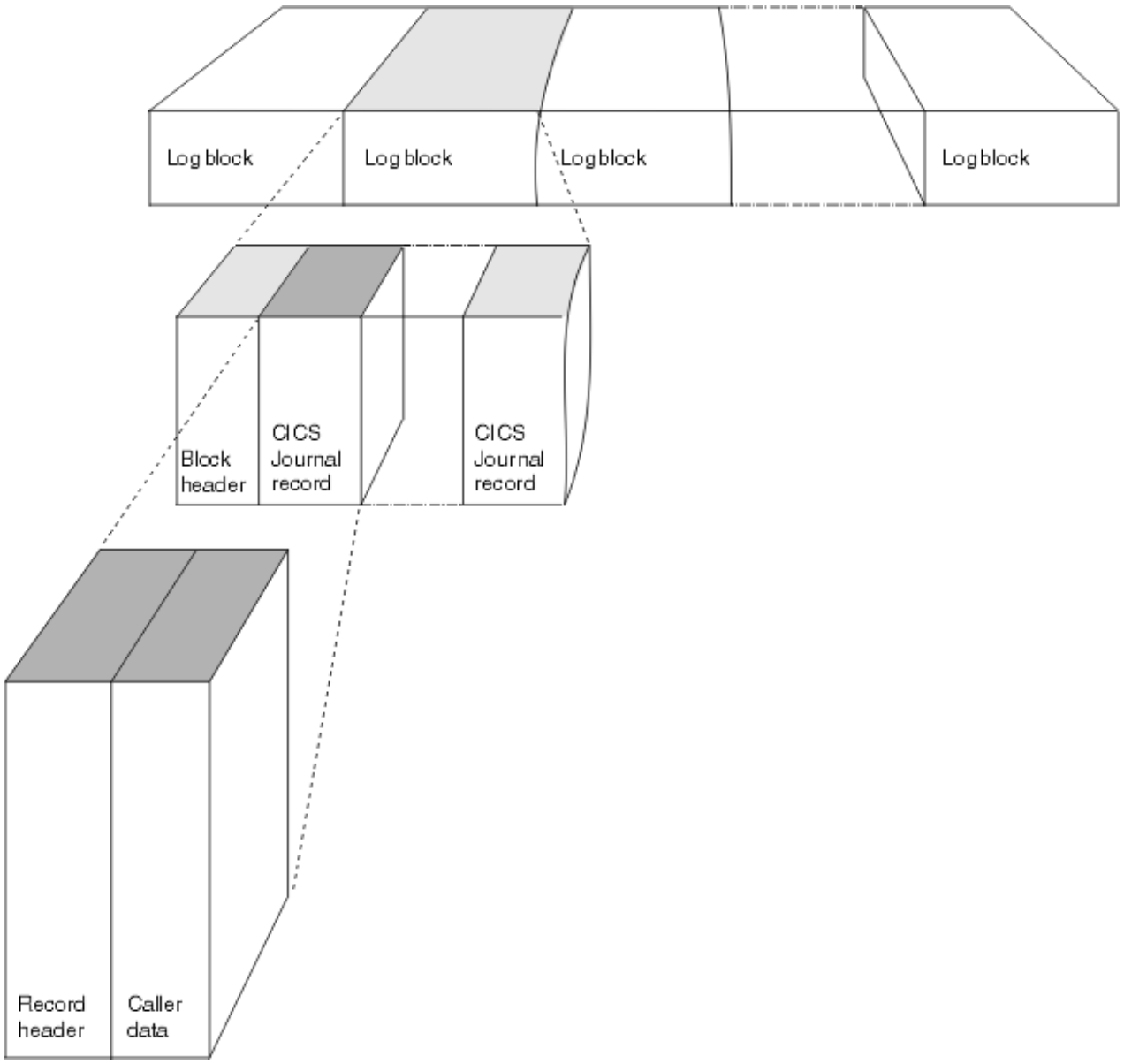


Figure 66. Layout of a general log

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 67 on page 273 shows the format of the log block header.

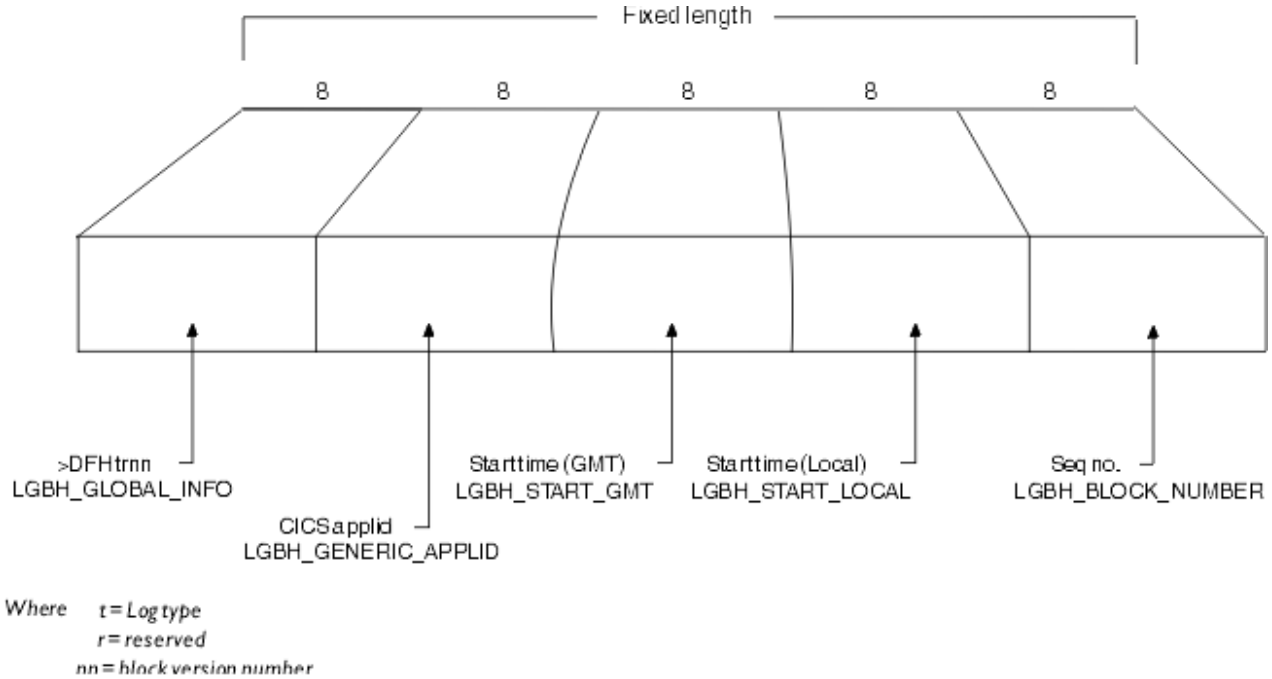


Figure 67. Format of a general log block header

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.

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. The caller data differs depending on the CICS component issuing the record, and on the function being journaled.

Figure 68 on page 274 shows the format of the record header.

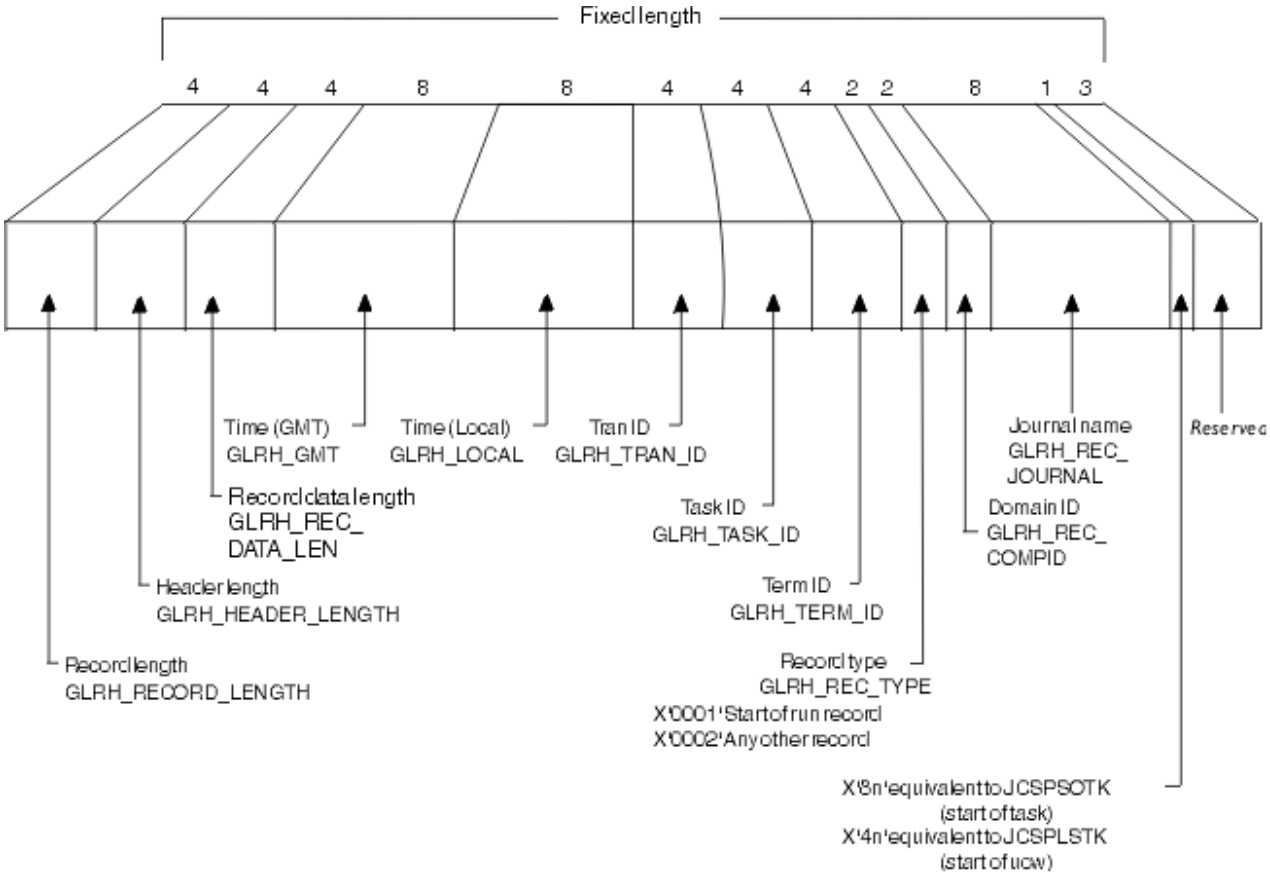


Figure 68. Format of a general log record header

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.

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 69 on page 275.

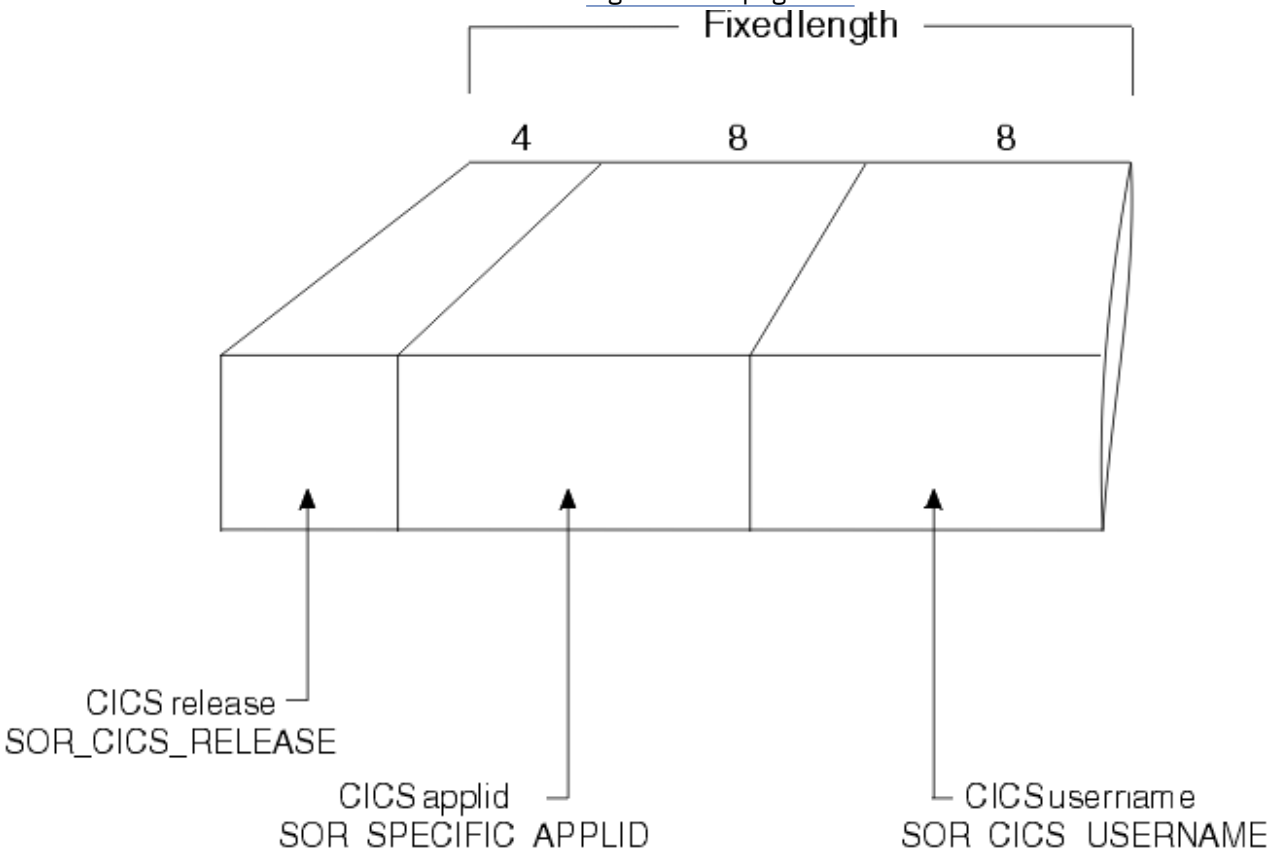


Figure 69. Format of the start-of-run record

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.

The 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 277. The other components (journal control, FEPI, and terminal control) do not add any further information to the journaled data.

The API user header

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 70 on page 276.

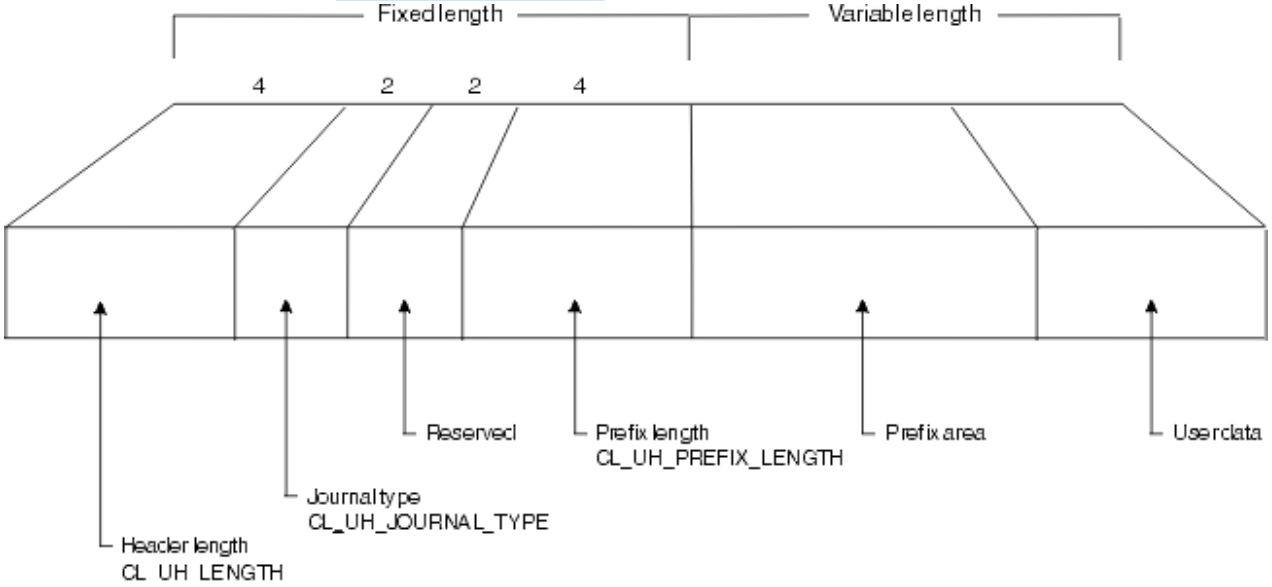


Figure 70. Format of the API user header

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 25 on page 277 outlines the sections in a journal record written by file control.

Table 25. 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 record types" on page 277	FLJB_GENERAL_ DATA	FLJB_COMMON_DATA	FLJB_CD_KEY	FLJB_CD_DATA
"Write-delete record types" on page 281	FLJB_GENERAL_ DATA	FLJB_WRITE_DELETE _DATA	FLJB_WDD_BASE _KEY	FLJB_WDD_PATH _KEY
"Commit and backout record types" on page 283	FLJB_GENERAL _DATA	None	None	None
"Unlock record types" on page 283	FLJB_GENERAL_ DATA	FLJB_UNLOCK_ DATA	FLJB_UND_BASE _KEY	FLJB_UND_PATH_KE Y
"File-close record types" on page 285	FLJB_GENERAL_ DATA	FLJB_FILE_CLOSE _DATA	None	None
"Tie-up record types" on page 286	FLJB_GENERAL_ DATA	FLJB_TIE_UP_RECORD _DATA	None	None

Read-only, read-update, write-update, write-add, write-add complete record types

The journal records written for read-only, read-update, write-update, write-add, and write-add complete record types consist of four sections.

These sections are as follows:

- The FLJB_GENERAL_DATA section

- The FLJB_COMMON_DATA section
- The two caller data image sections:
  - FLJB_CD_KEY (its length is given in FLJB_COMMON_DATA)
  - FLJB_CD_DATA (its length is given in FLJB_COMMON_DATA)

    This section contains the image of the caller data.

Record format

The format of such a record written for these record types is shown in Figure 71 on page 278.

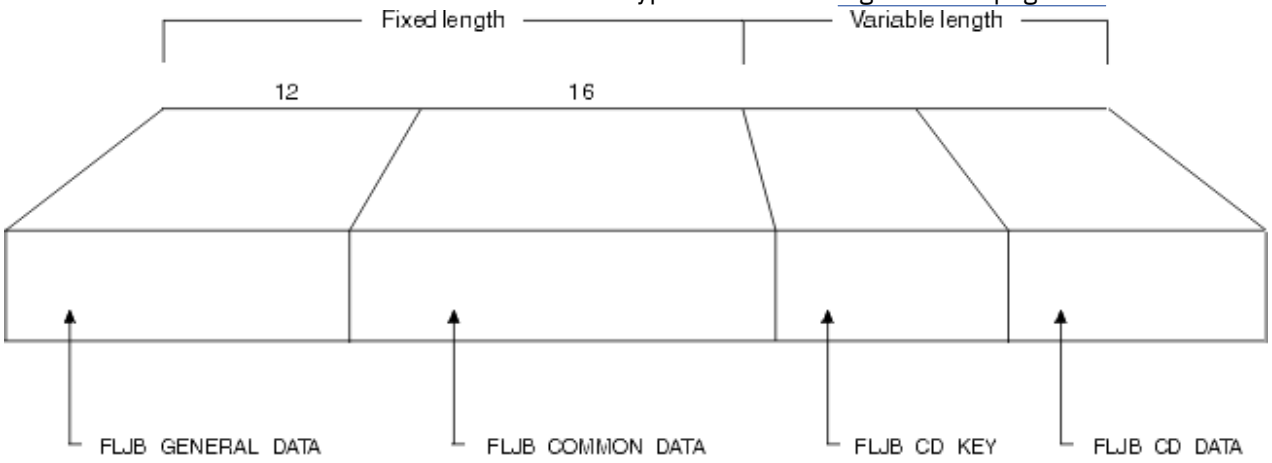


Figure 71. Layout of record written for read-only, read-update, write-update, write-add, and write-add-complete record types

FLJB_GENERAL_DATA
12-byte general data.

See “FLJB_GENERAL_DATA” on page 278.

FLJB_COMMON_DATA
16-byte common data.

See “FLJB_COMMON_DATA” on page 280.

FLJB_CD_KEY
Variable-length caller data key.

FLJB_CD_DATA
Variable-length caller data.

FLJB_GENERAL_DATA

The format of the FLJB_GENERAL_DATA section is shown in Figure 72 on page 279.

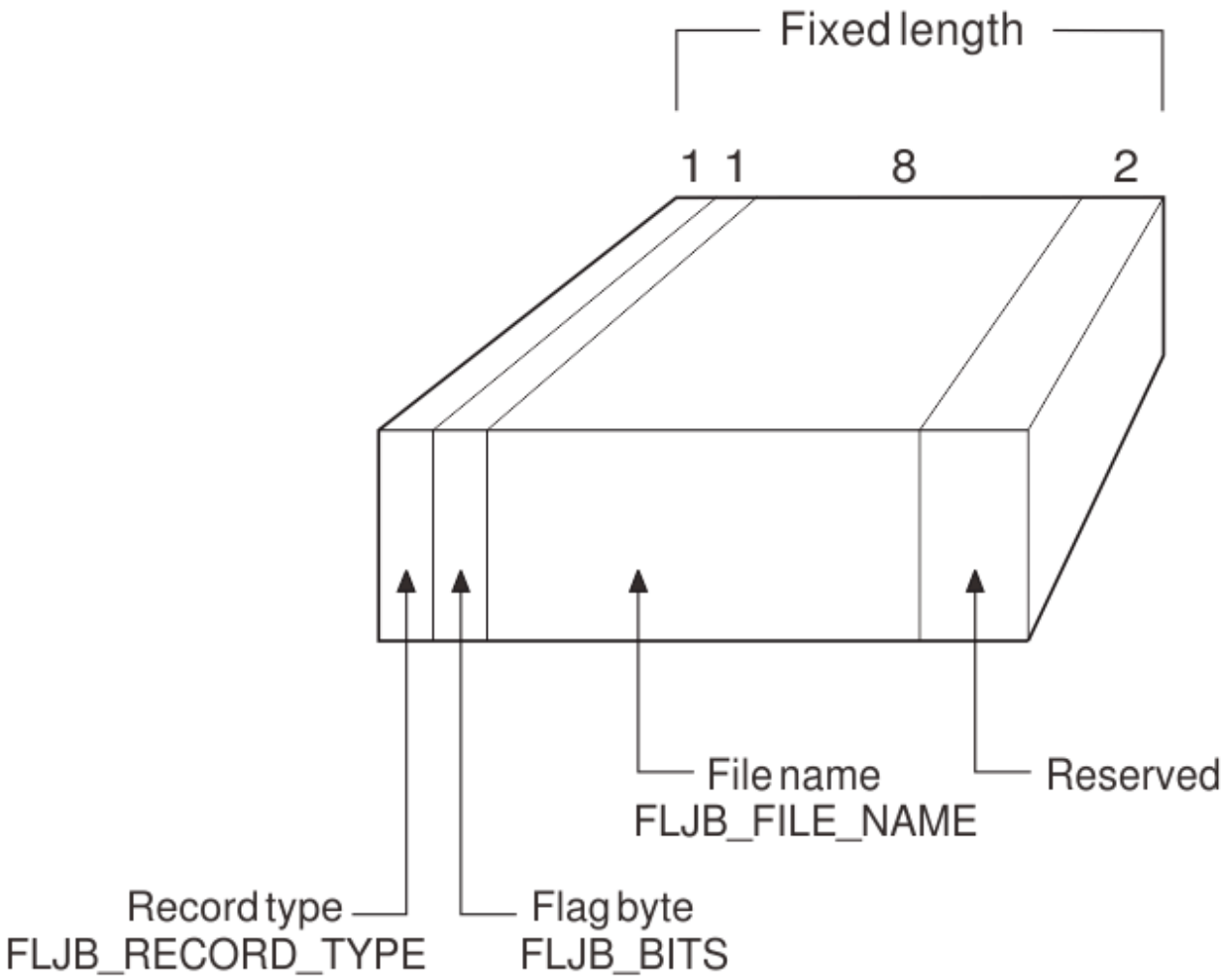


Figure 72. Format of FLJB_GENERAL_DATA section

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'87'    Commit (replication only)
X'88'    Backout (replication only)
X'89'    Unlock (replication only)
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
X'02'    Replication record
X'01'    Replication record written by replication tran
```

FLJB_FILE_NAME
8-byte file name.

Reserved
2-byte reserved field.

FLJB_COMMON_DATA

The format of the FLJB_COMMON_DATA section is shown in Figure 73 on page 280.

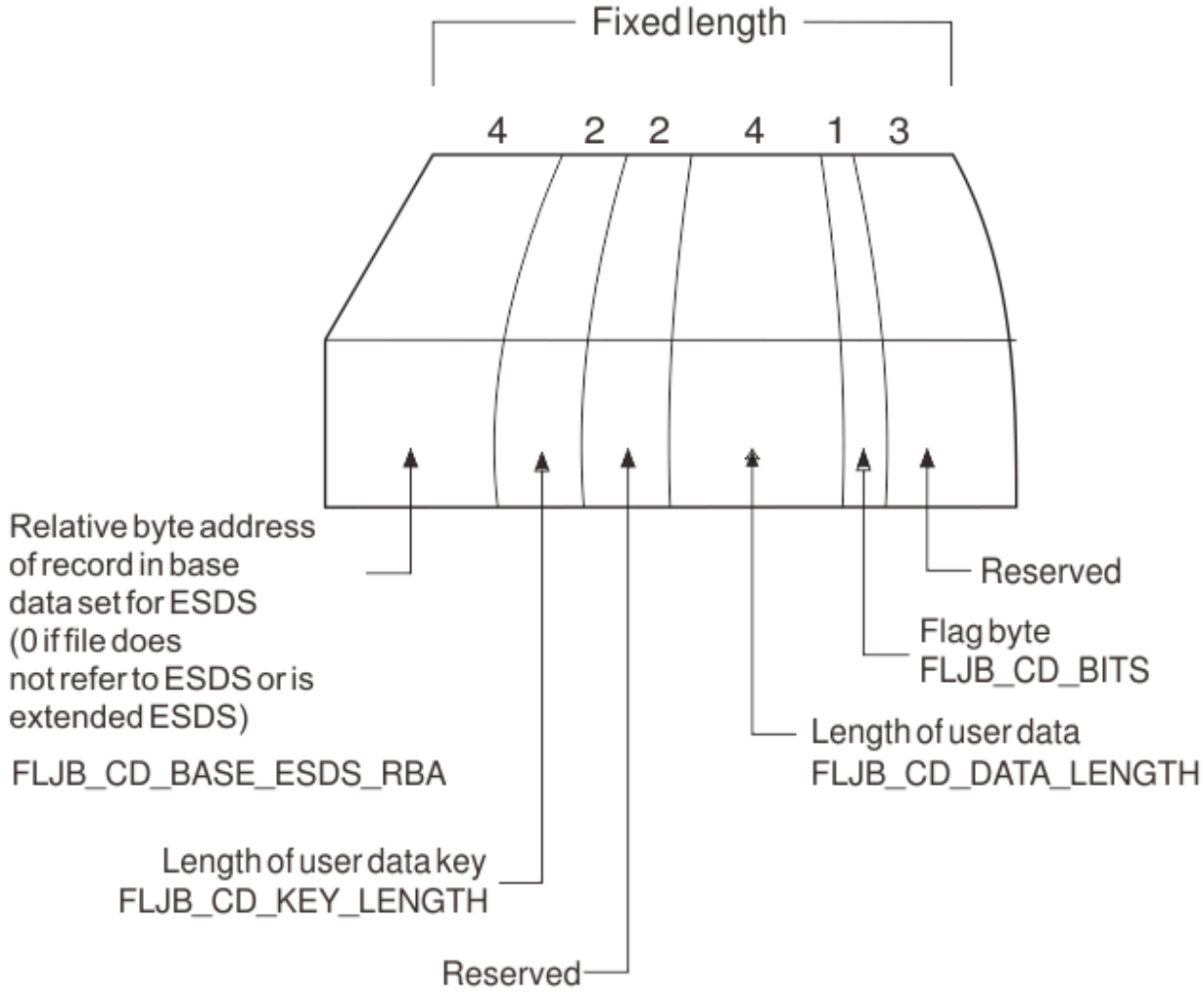


Figure 73. Format of FLJB_COMMON_DATA section

FLJB_CD_BASE_ESDS_RBA
4-byte relative byte address of record in the base data set for an ESDS.

The relative byte address is 0 if the file does not refer to an ESDS, or if it 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; it is 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'08'    Fixed length record
X'04'    Replication record is auto committed
X'02'    Token used on READ-UPDATE (replication records only)
```

Combinations of settings are possible.

Reserved
3-byte reserved field.

Write-delete record types

The journal records written for write-delete record types consist of four sections.

These sections are as follows:

- The FLJB_GENERAL_DATA section
- The FLJB_WRITE_DELETE_DATA section
- The two caller data image sections:
  - FLJB_WDD_BASE_KEY (its length is given by FLJB_WDD_BASE_KEY_LENGTH in FLJB_WRITE_DELETE_DATA)
  - FLJB_WDD_PATH_KEY (its length 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.

Record format for a write-delete record

The format of such a record written for write-delete record types is shown in Figure 74 on page 281.

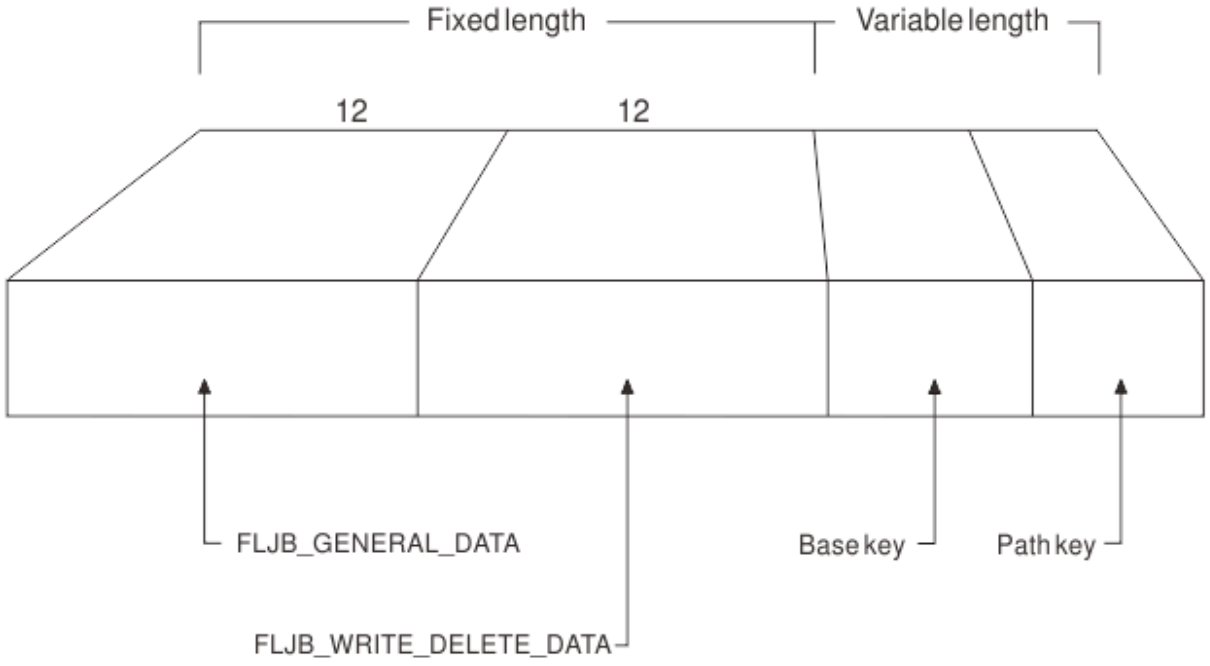


Figure 74. Layout of record written for write-delete record types

FLJB_GENERAL_DATA
12-byte general data section.

See “FLJB_GENERAL_DATA” on page 278.

FLJB_WRITE_DELETE_DATA
12-byte write-delete data section.

See “FLJB_WRITE_DELETE_DATA” on page 282.

FLJB_WDD_BASE_KEY
Variable-length base key.

FLJB_WDD_PATH_KEY
Variable-length path key.

FLJB_WRITE_DELETE_DATA

The format of the FLJB_WRITE_DELETE_DATA section is shown in Figure 75 on page 282.

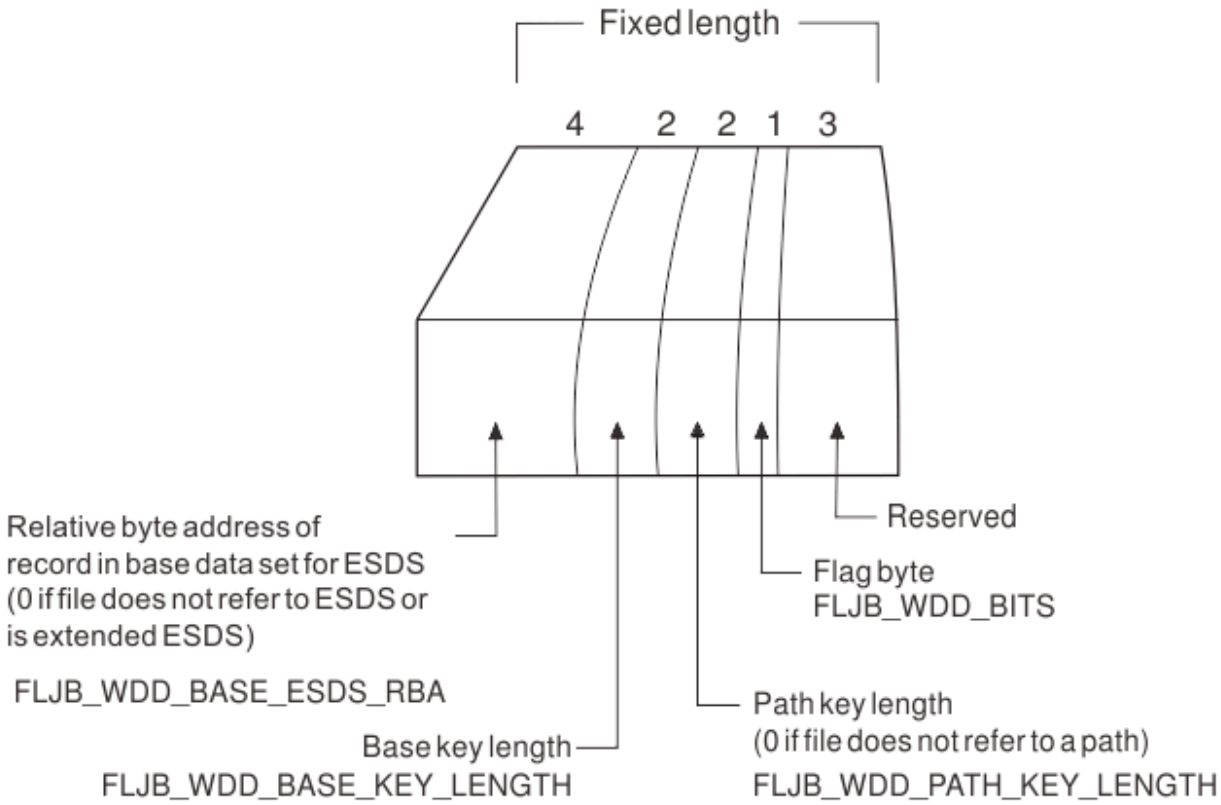


Figure 75. Format of the FLJB_WRITE_DELETE_DATA section

FLJB_WDD_BASE_ESDS_RBA
4-byte relative byte address of record in base data set for ESDS.

The relative byte address is 0 if the file does not refer to an 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; it is 8 for an extended addressing ESDS.

FLJB_WDD_PATH_KEY_LENGTH
2-byte path key length.

The key length is 0 if the 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.

Commit and backout record types

The journal records written for commit and backout are written for replication logging only and consist of one section.

- The FLJB_GENERAL_DATA section

FLJB_GENERAL_DATA
12-byte general data section.

See "FLJB_GENERAL_DATA" on page 278.

Unlock record types

The journal records written for unlock are written for replication logging only and consist of four sections.

These sections are as follows:

- The FLJB_GENERAL_DATA section
- The FLJB_UNLOCK_DATA section
- The two caller data image sections:
  - FLJB_UND_BASE_KEY (its length is given by FLJB_UND_BASE_KEY_LENGTH in FLJB_UNLOCK_DELETE_DATA)
  - FLJB_UND_PATH_KEY (its length is given by FLJB_UND_PATH_KEY_LENGTH in FLJB_UNLOCK_DATA)

  These sections contain the image of the caller data as the base key, and, if the data set is a path, the path.

Record format for an unlock record

The format of a record written for unlock record types is shown in Figure 76 on page 283.

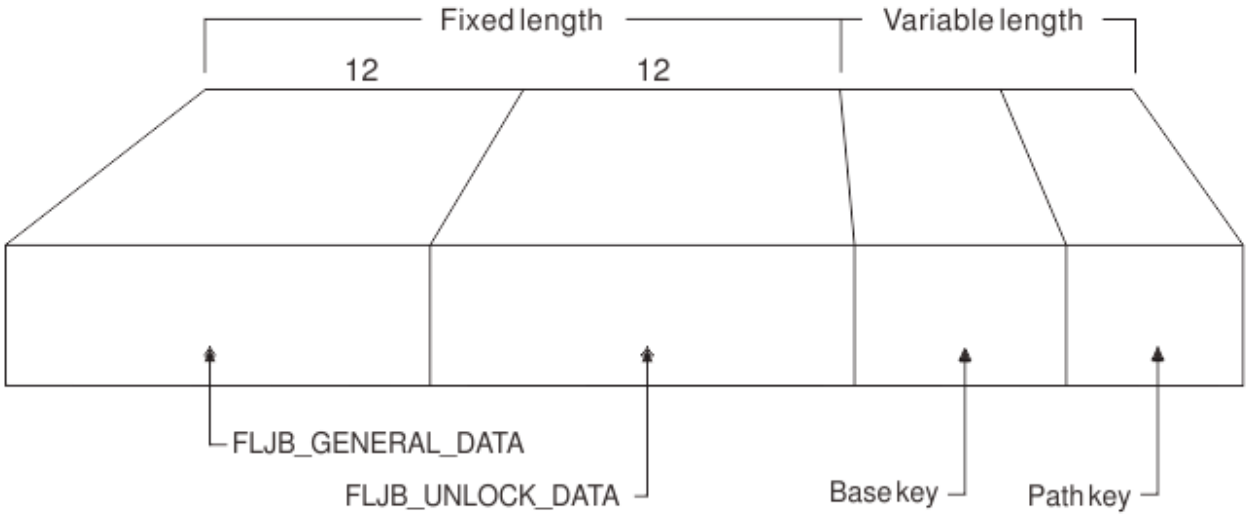


Figure 76. Layout of record written for unlock record types

FLJB_GENERAL_DATA
12-byte general data section.

See “FLJB_GENERAL_DATA” on page 278.

FLJB_UNLOCK_DATA
12-byte unlock data section.

See “FLJB_UNLOCK_DATA” on page 284.

FLJB_UND_BASE_KEY
Variable-length base key.

FLJB_UND_PATH_KEY
Variable-length path key.

FLJB_UNLOCK_DATA

The format of the FLJB_UNLOCK_DATA section is shown in Figure 77 on page 284.

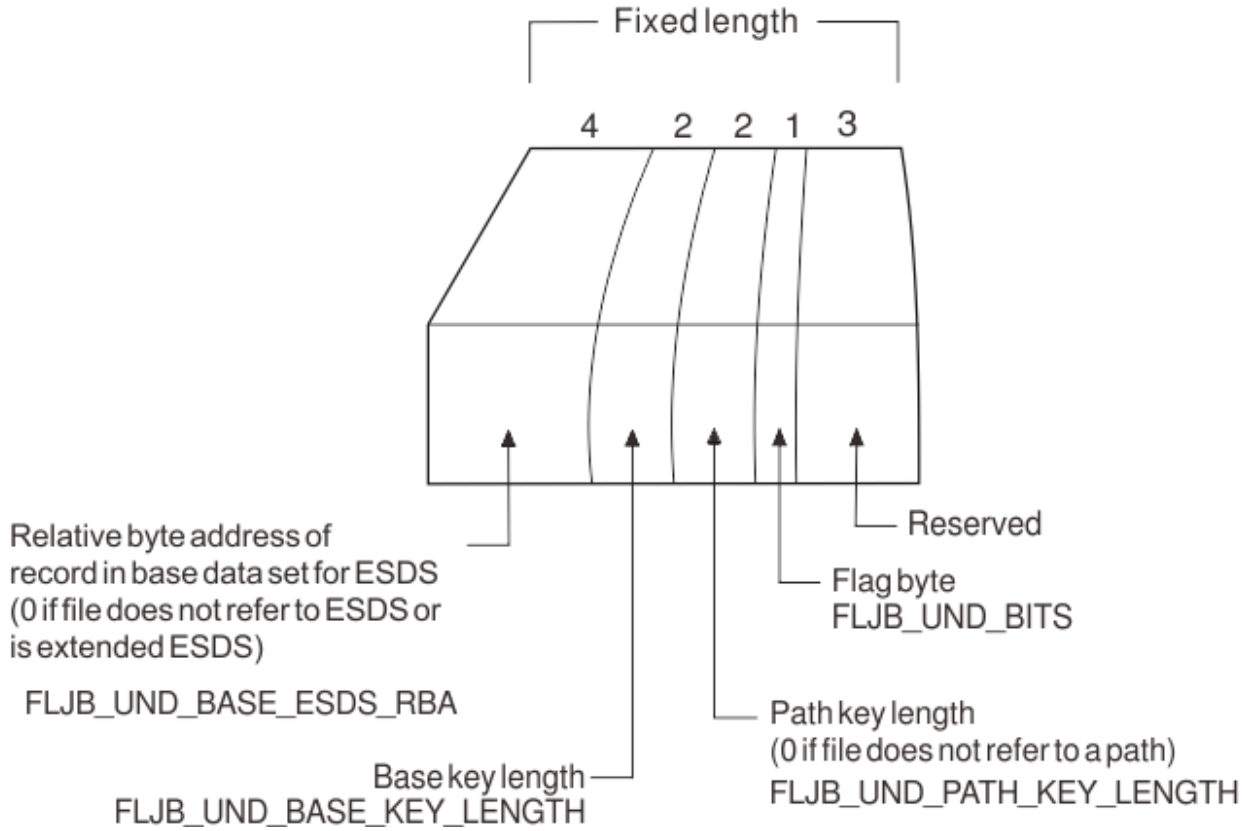


Figure 77. Format of the FLJB_UNLOCK_DATA section

FLJB_UND_BASE_ESDS_RBA
4-byte relative byte address of record in the base data set for an ESDS.

The relative byte address is 0 if the file does not refer to an ESDS, or if it is an extended addressing ESDS.

FLJB_UND_BASE_KEY_LENGTH
2-byte base key length.

The key length is 4 for an RRDS, a VRRDS, or a standard ESDS; it is 8 for an extended addressing ESDS.

FLJB_UND_PATH_KEY_LENGTH
2-byte path key length.

The key length is 0 if the file does not refer to a path.

FLJB_UND_BITS
1-byte flag field:

```
X'80'    UOW has been shunted at least once
X'40'    Fixed-length record
X'20'    Replication record is auto committed
X'10'    Unlock follows a read-update
X'08'    Unlock follows a massinsert
```

Reserved
3-byte reserved field.

File-close record types

The journal records written for file-close record types consist of two sections.

These sections are as follows:

- The FLJB_GENERAL_DATA section
- The FLJB_CLOSE_DATA section

Record format for a file-close record

The format of such a record written for file-close record types is shown in Figure 78 on page 285.

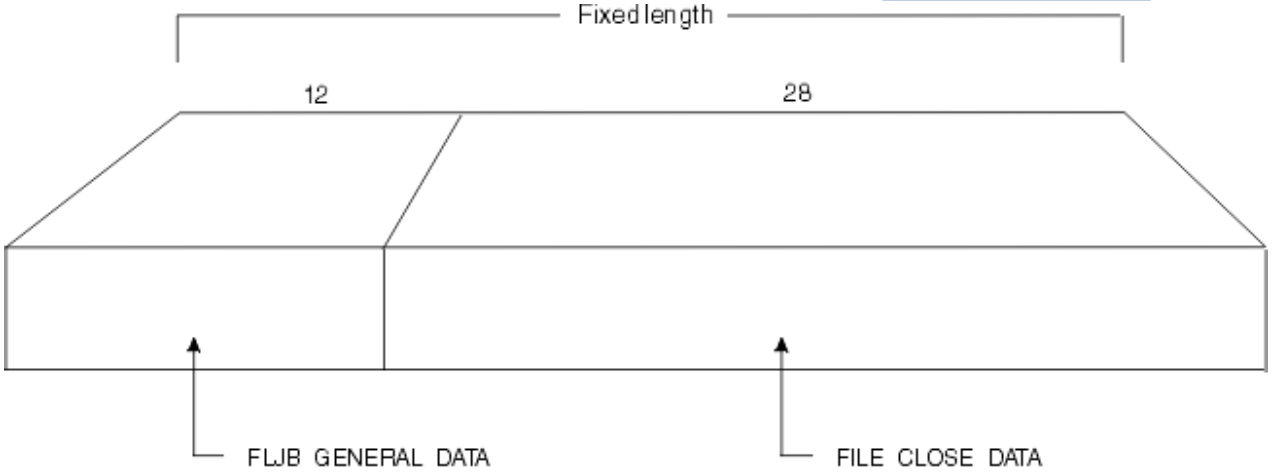


Figure 78. Layout of record written for file-close record types

FLJB_GENERAL_DATA
12-byte general data section.

See “FLJB_GENERAL_DATA” on page 278.

FLJB_CLOSE_DATA
28-byte close data section.

See “FLJB_CLOSE_DATA” on page 285.

FLJB_CLOSE_DATA

The format of the FLJB_CLOSE_DATA section is shown in Figure 79 on page 286.

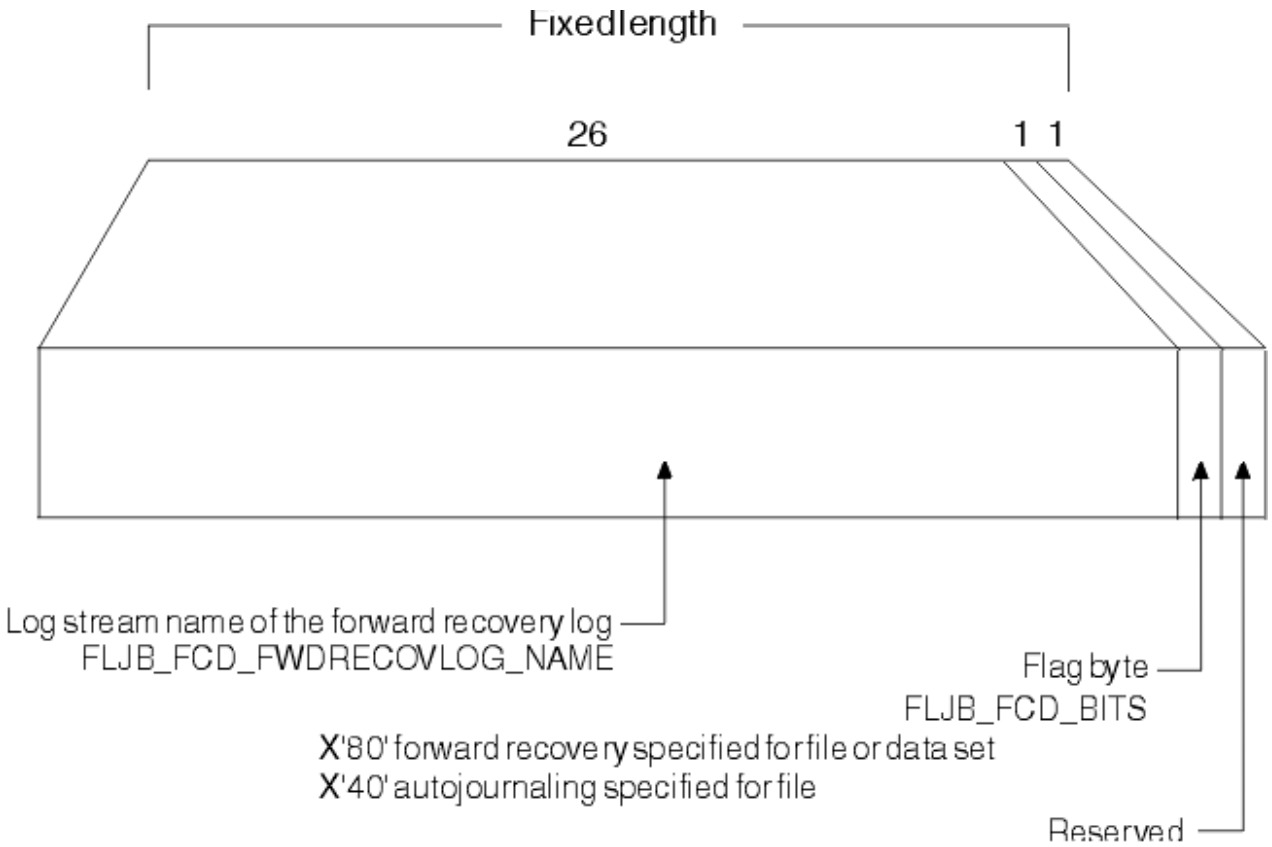


Figure 79. Format of the FILE_CLOSE_DATA section

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

The journal records written for tie-up record types consist of two sections.

These sections are as follows:

- The FLJB_GENERAL_DATA section
- The TIE_UP_RECORD_DATA section

Record format for a tie-up record

The format of such a record written for tie-up record types is shown in Figure 80 on page 287.

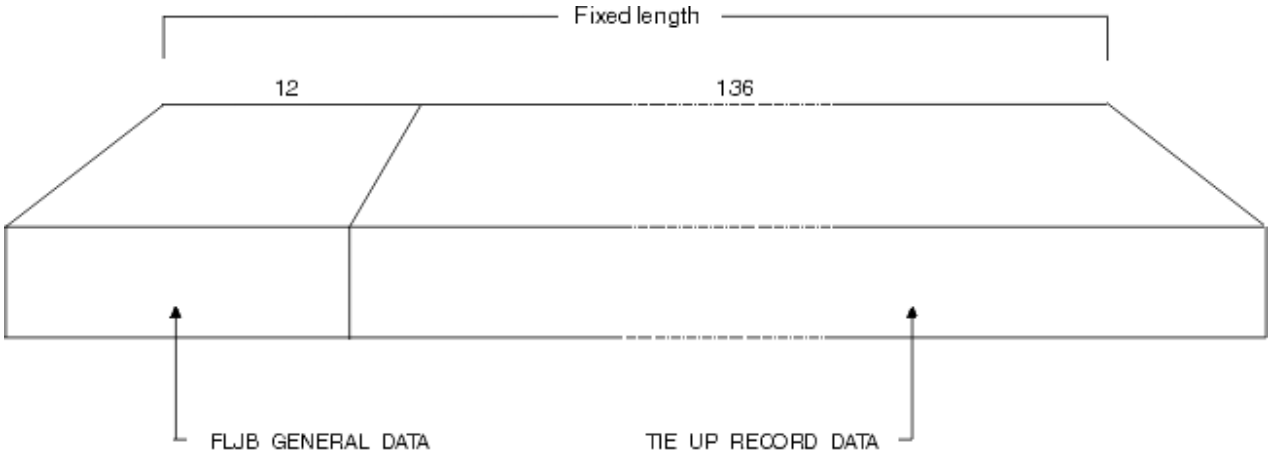


Figure 80. Layout of record written for tie-up record types

FLJB_GENERAL_DATA
12-byte general data.

See "FLJB_GENERAL_DATA" on page 278.

TIE_UP_RECORD_DATA
136-byte tie-up record data.

See "TIE_UP_RECORD_DATA" on page 287.

TIE_UP_RECORD_DATA

The format of the TIE_UP_RECORD_DATA section is shown in Figure 81 on page 288.

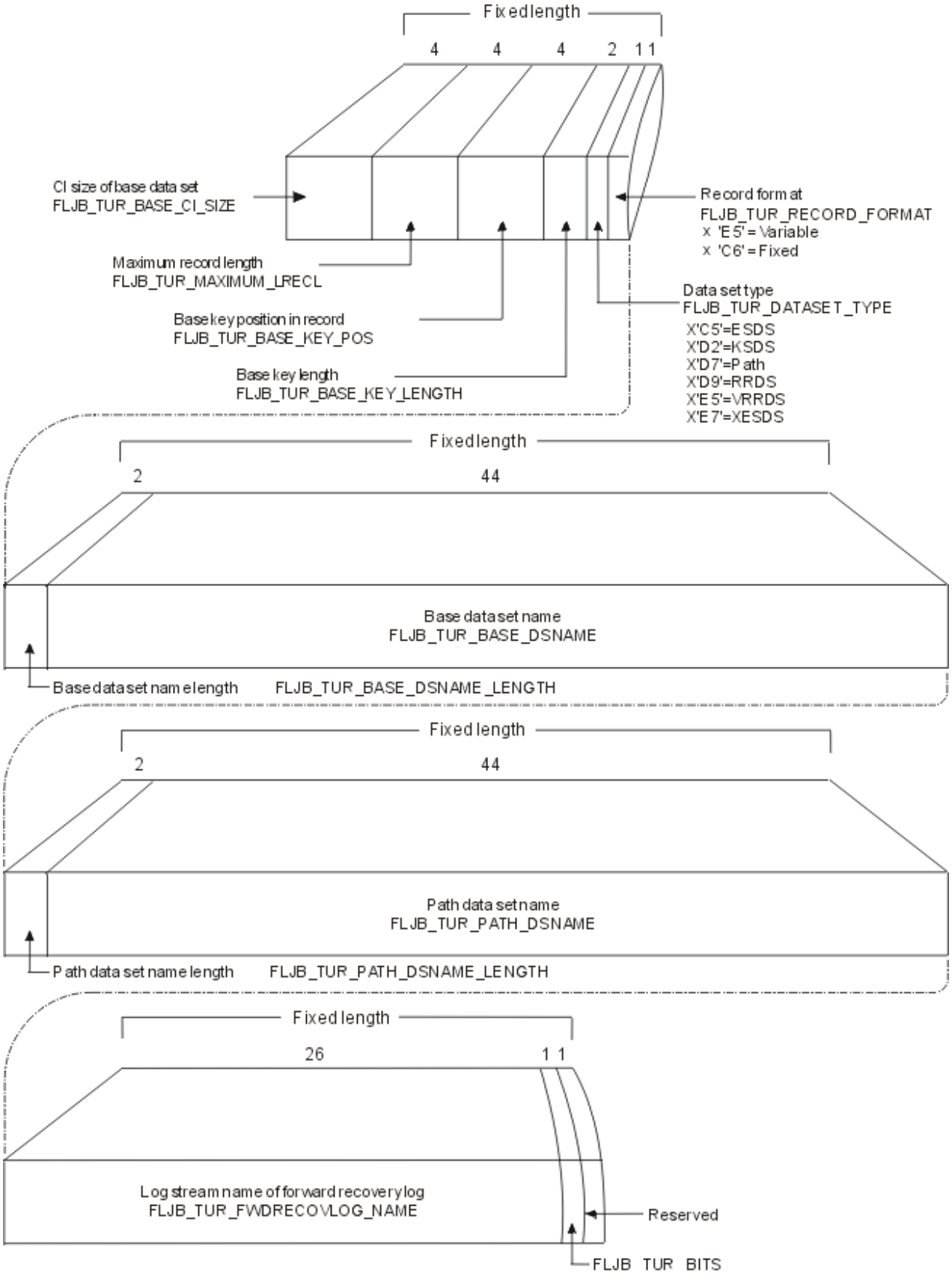


Figure 81. Format of TIE_UP_RECORD_DATA section

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 82 on page 290.

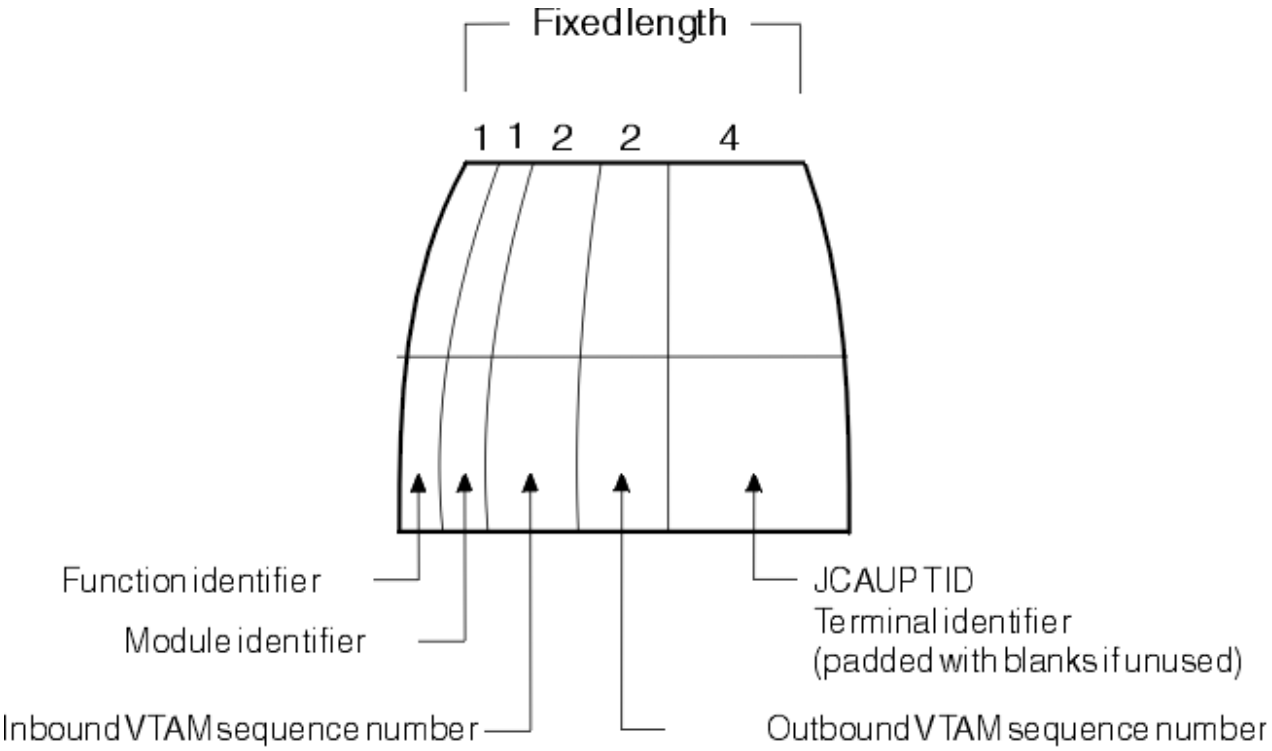


Figure 82. Format of the terminal control prefix area

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 83 on page 291.

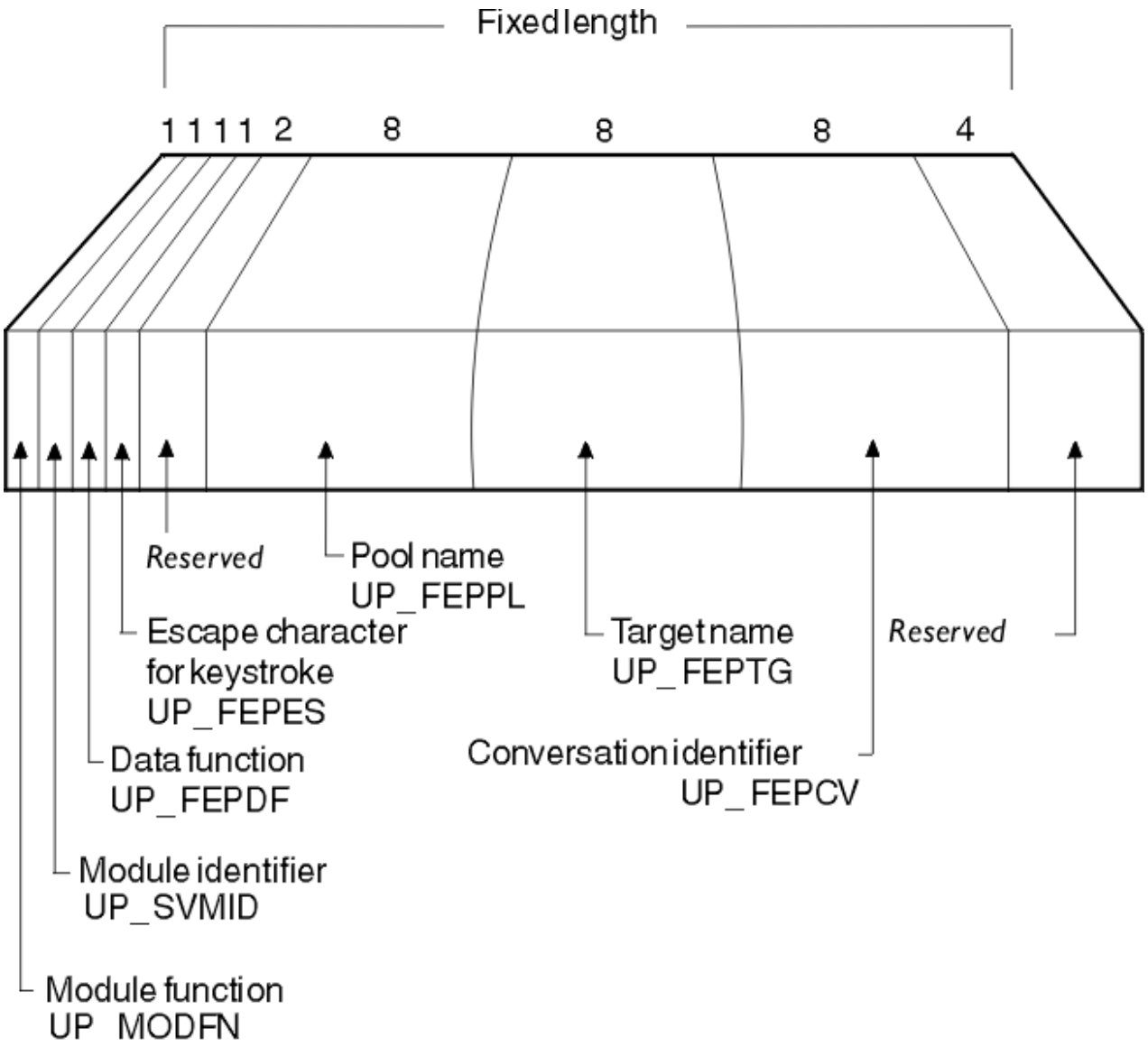


Figure 83. Format of the FEPI prefix area

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

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.

Note:

For **SMF records**, see “Format of journal records written to SMF” on page 303.

This section does not apply to journal records written to an SMF data set.

Fields that are not presented within the data

Within the data, certain fields as shown in Table 26 on page 292 are not presented. They appear as X'00' in the formatted output.

Table 26. Fields formatted as X'00'

Field
JCLRJFID
JCLRTBAL
JCSPEMER
JCLRVCD
JCRBB
JCSPMIDT
JCLRVSN
JCSPFS
JCSPRRIF
JCLRLBW
JCSPDSP

Format of a journal record

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 84 on page 293.

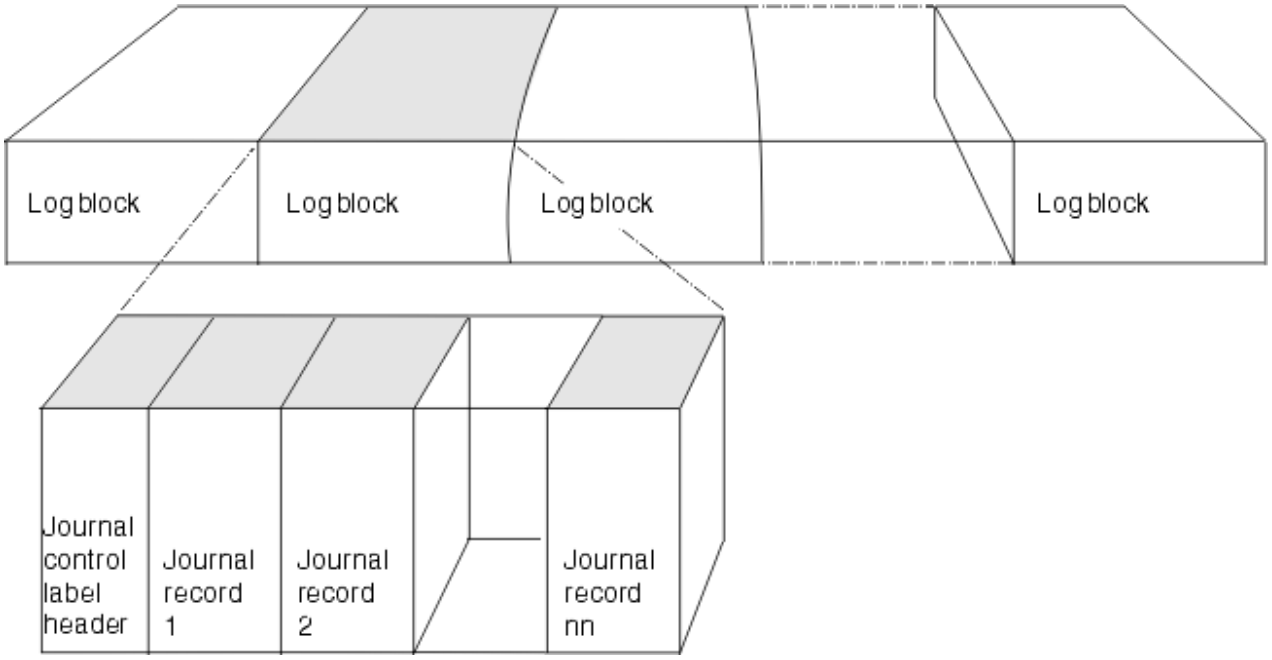


Figure 84. Format of general log formatted using the COMPAT41 option

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 85 on page 293.

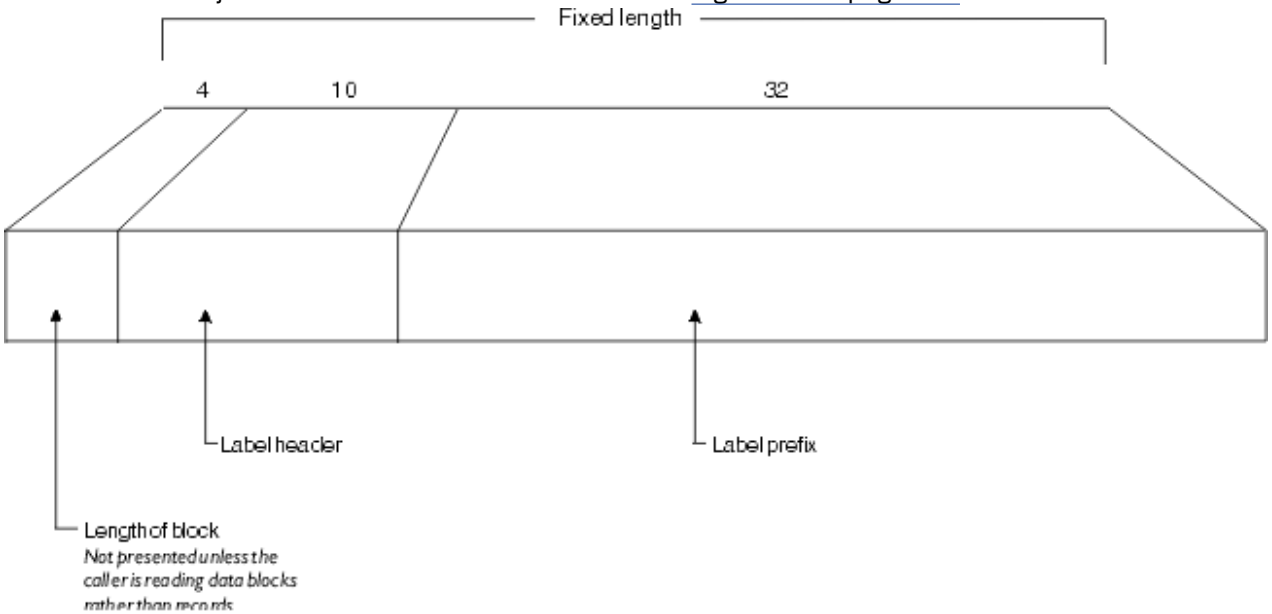


Figure 85. Format of journal control label header

Label header part

The label header part of the journal control label header is 10 bytes long, and its format is shown in Figure 86 on page 294.

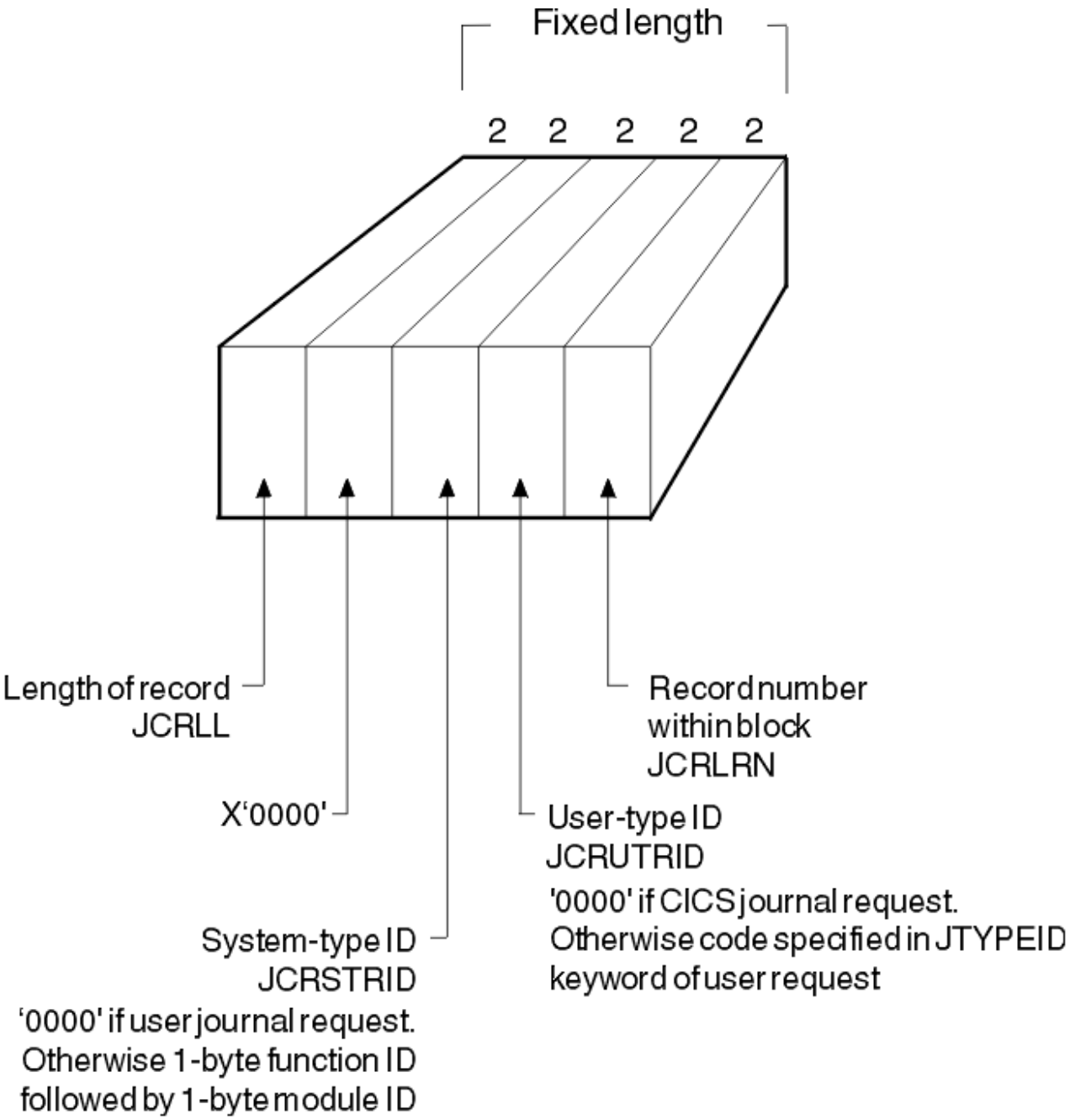


Figure 86. Format of label header part of journal control label header

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.

Label prefix part

The label prefix part of the journal control label header is 32 bytes long, and its format is shown in Figure 87 on page 295.

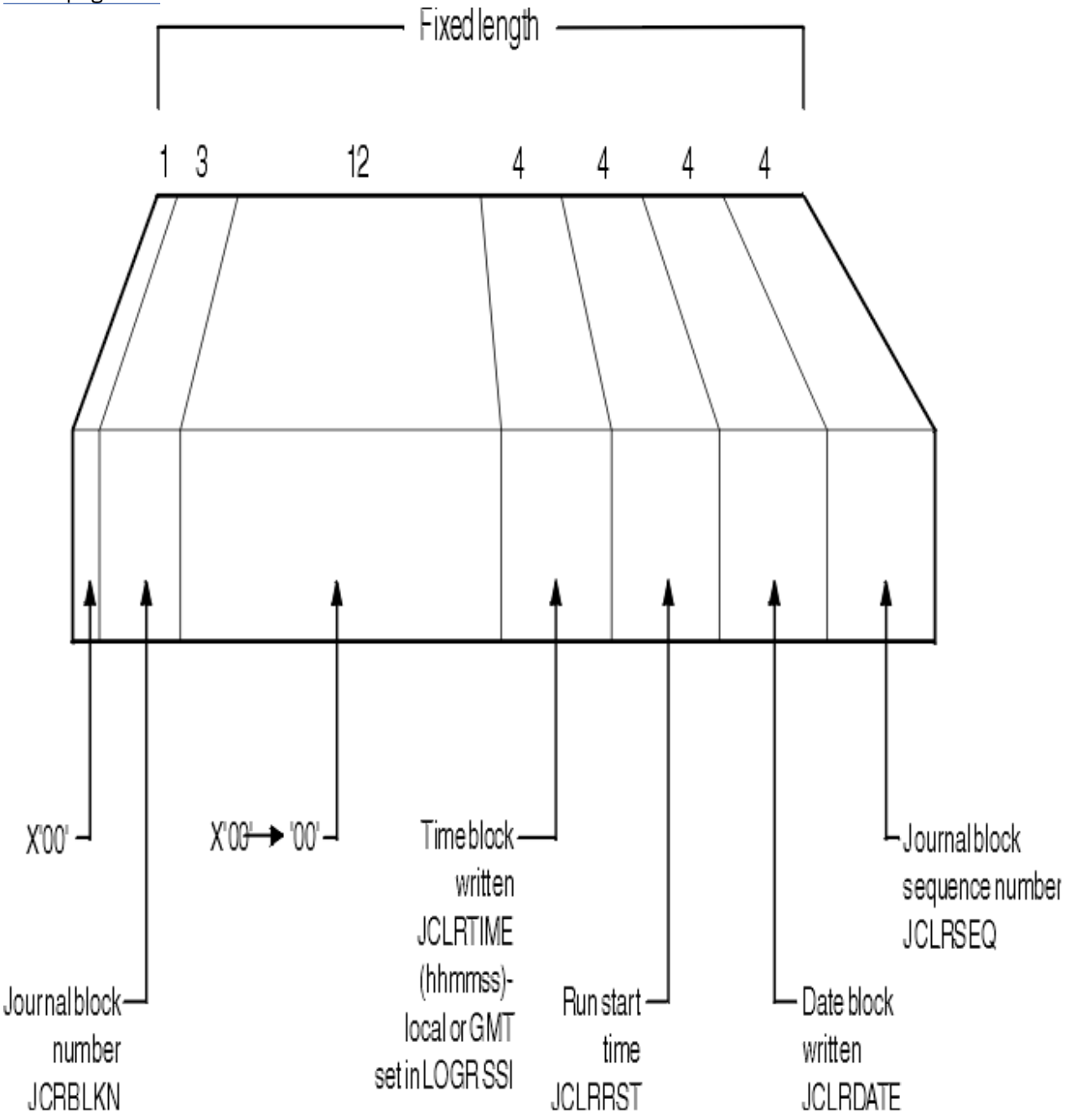


Figure 87. Format of label prefix part of journal control label header

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.

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 88 on page 296.

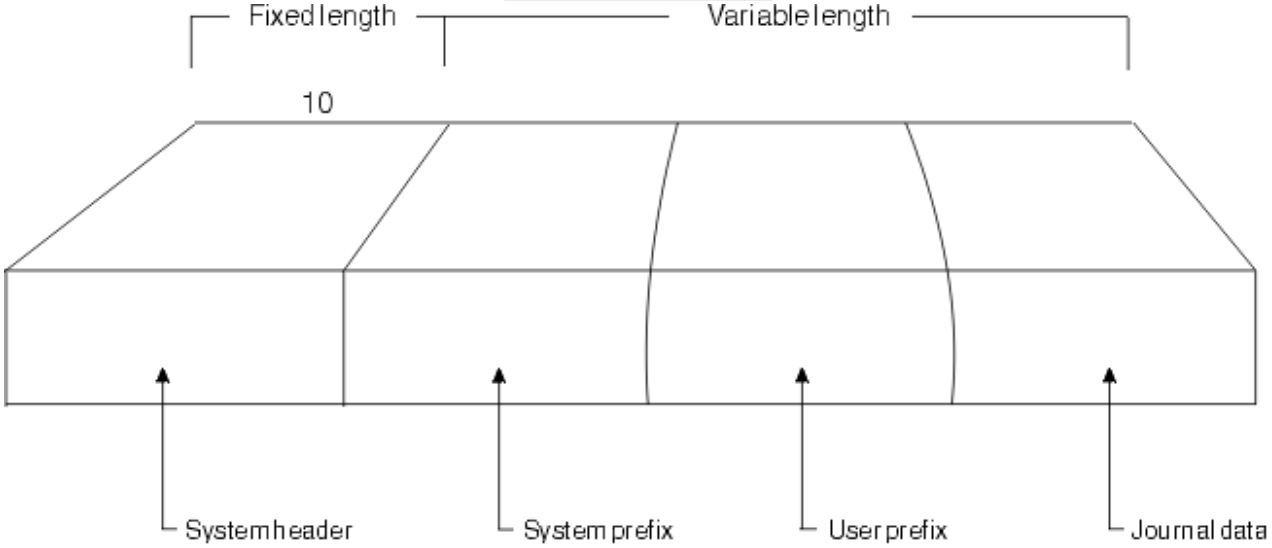


Figure 88. Format of COMPAT41 journal record

The system header is 10 bytes long. Its format is shown in Figure 89 on page 297.

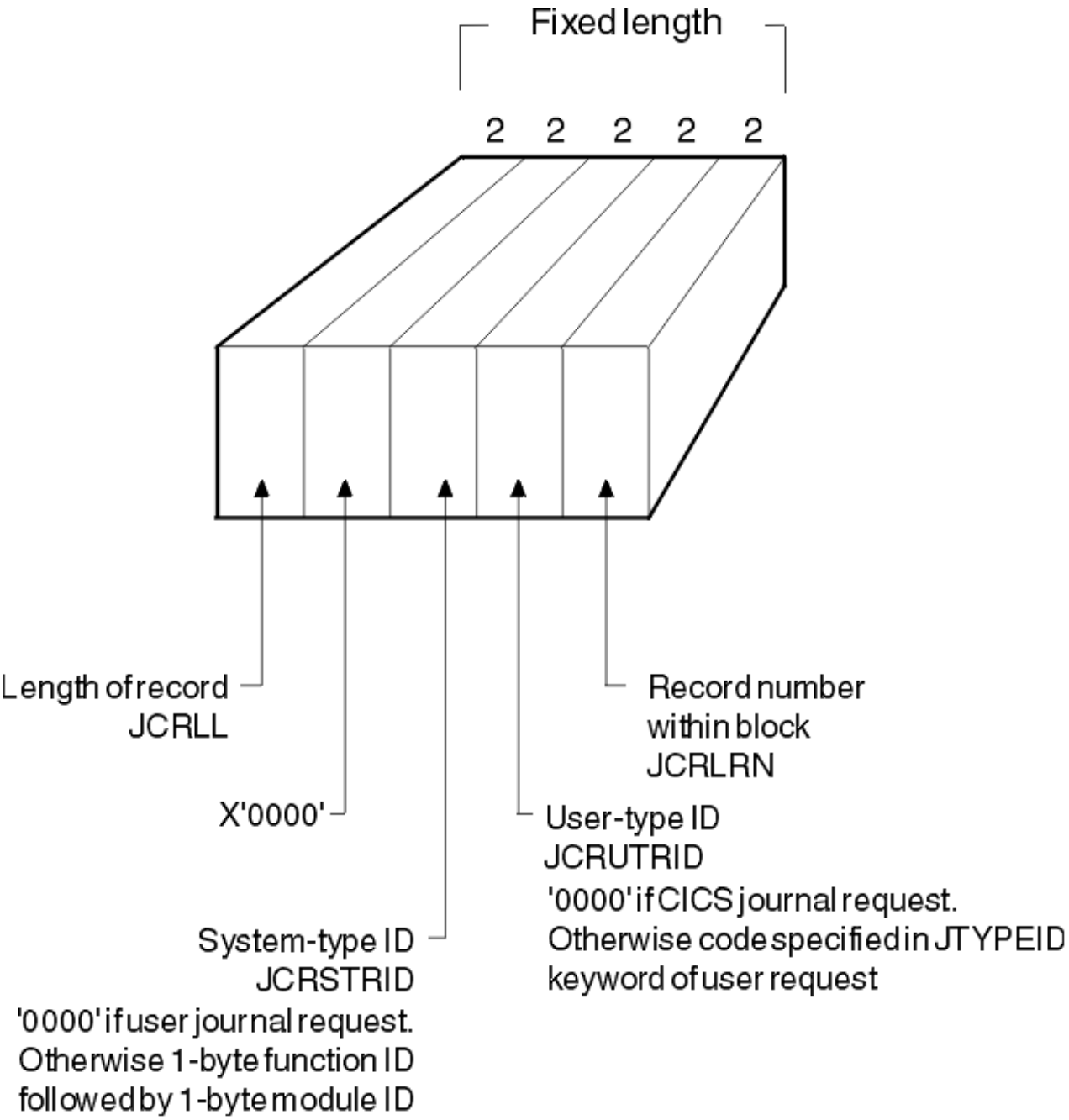


Figure 89. Format of the system header

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 92 on page 301 shows the valid function identifiers of those CICS components that issue journal requests. Figure 94 on page 302 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 90 on page 298.

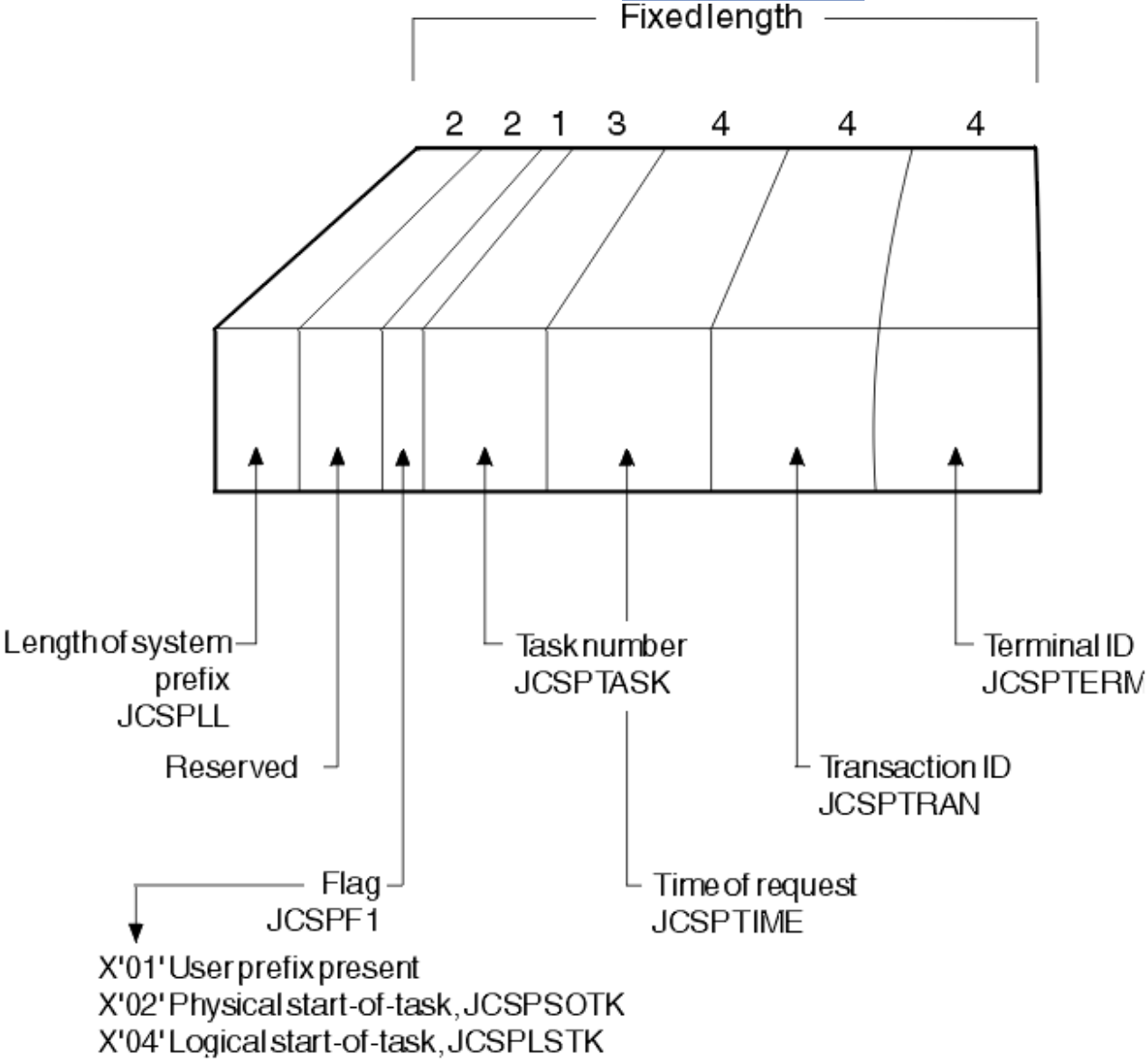


Figure 90. Format of the system prefix

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 Data areas.

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 91 on page 300.

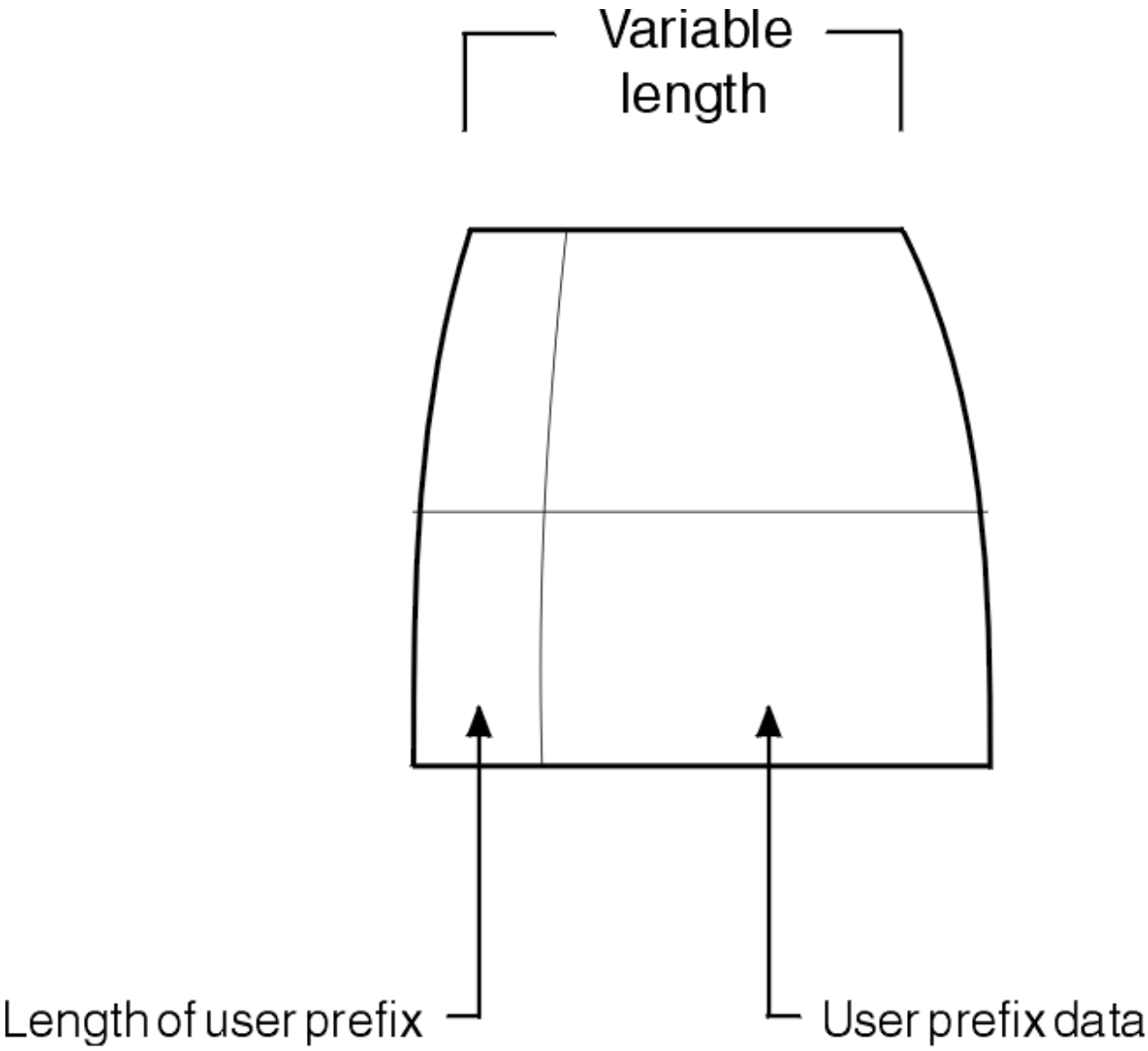


Figure 91. Format of the user prefix

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.
```

Figure 92. Journal function identifiers (part 1)

```
*************************************************************************
*               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
FIDTCIMN EQU   X'8B'               INPUT MSG, NO FMH, COMPLETE
FIDTCINN EQU   X'8C'               INPUT MSG, NO FMH, INCOMPLETE
*************************************************************************
*             FRONT END PROGRAMMING INTERFACE IDENTIFIERS               *
FIDFEPIN EQU   X'F0'               FEPI INBOUND DATA    API <--- FEPI
FIDFEPOU EQU   X'F1'               FEPI OUTBOUND DATA   API ---> FEPI
*************************************************************************
```

Figure 93. Journal function identifiers (part 2)

```
*************************************************************************
* *              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 94. 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).

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.

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 95 on page 303.

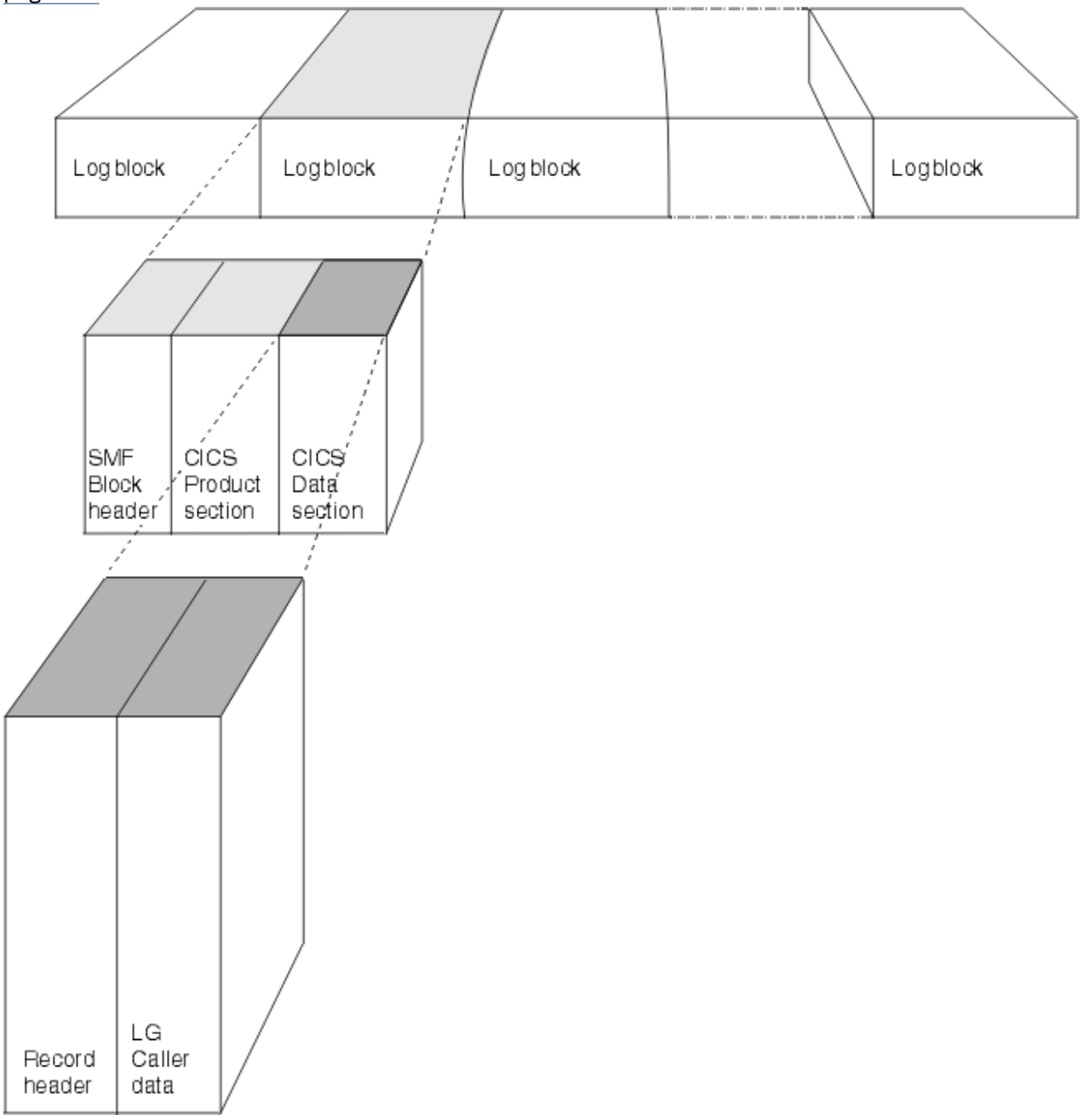


Figure 95. Layout of a CICS log written to MVS SMF

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.

The SMF block header

The SMF block header describes the system creating the output.

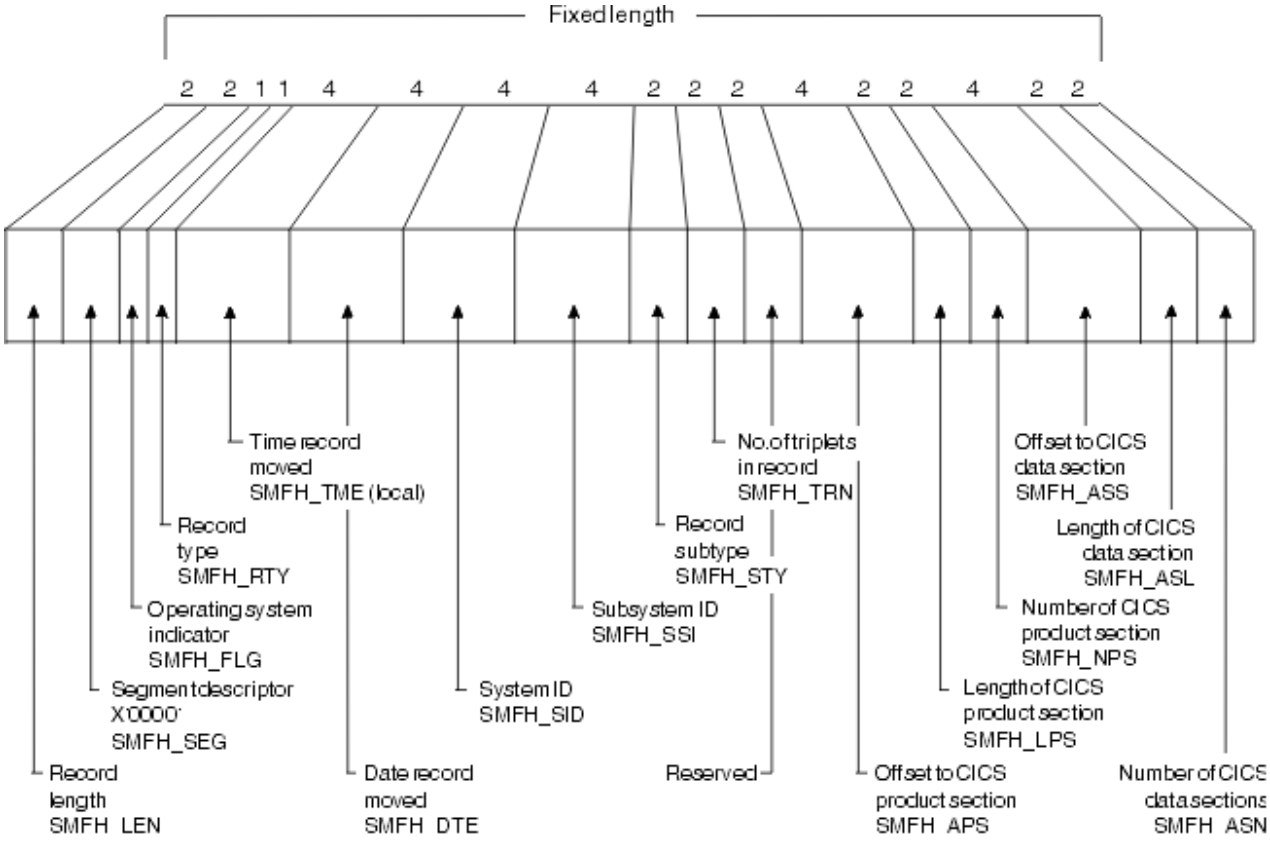


Figure 96. Format of the SMF block header

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,see z/OS MVS System Management Facilities (SMF).

The CICS product section

The CICS product section identifies the subsystem to which the journaling data relates.

Its format is shown in Figure 97 on page 306.

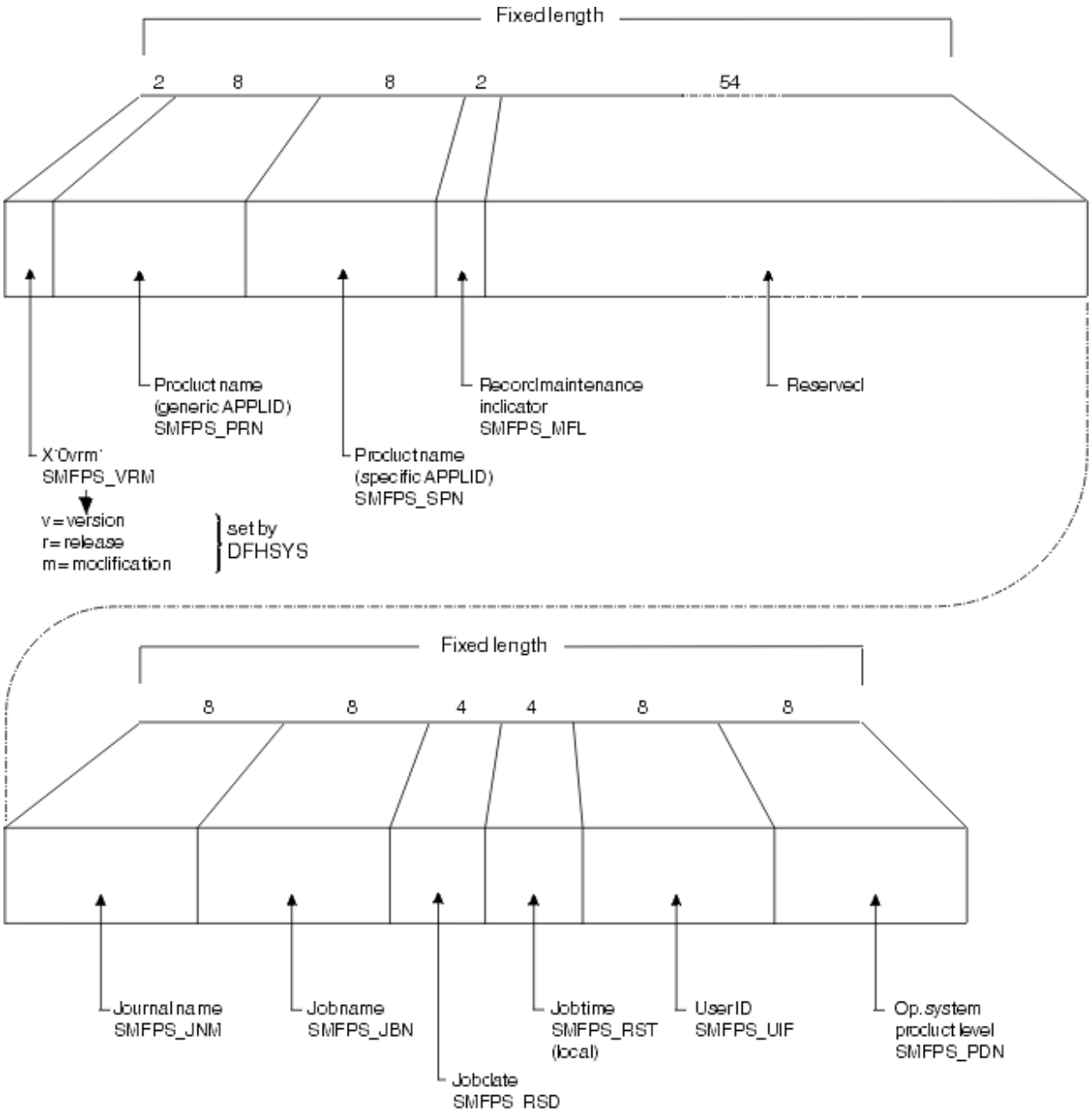


Figure 97. Format of the CICS product section

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.

The CICS data section

This section contains a variable number of CICS journal records.

Each record starts with a general log record header, which is followed by a user header. The user header 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. Subsequent records then take the form already described in the preceding paragraph.

Chapter 5. 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.

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 CICSTS55.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** main terminal transaction, described in CEMT SET FILE, 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.

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.

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.

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.

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.

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 CICSTS55.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.

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.

Chapter 6. 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 z/Architecture Principles of Operation.

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 data sets.

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.

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.

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).

For an overview of DFHCSDUP, see System definition file utility program (DFHCSDUP).

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 DFHSTUP extract statistics reporting function.

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 z/Architecture Principles of Operation.) 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 317.

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 27. 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$CRFA and DFH0CRFC respectively, are supplied in executable form in CICSTS55.CICS.SDFHLOAD. The PL/I version, DFH$CRFP, is supplied in source form only.

The assembler-language and COBOL versions of the Db2 formatting program, DFH$FORA and DFH0FORC respectively, are supplied in executable form in CICSTS55.CICS.SDFHLOAD. The PL/I version, DFH$FORP, is supplied in source form only.

The CSD backup utility program, DFH0CBDC, is supplied in executable form in CICSTS55.CICS.SDFHLOAD.

The source statements of all versions of all the sample programs are supplied in CICSTS55.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 5 ; if the option is not specified, only keywords from CICS Transaction Server for z/OS, Version 5 Release 5 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 CSD cross-referencing program

Use the CICS-supplied sample CSD cross-referencing program to produce 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 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=CICSTS55.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 98. 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 99 on page 323 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=CICSTS55.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 99. 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 329), 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. (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.

The CICS-supplied sample program, DFH$CUS1, illustrates how the DFHCSDUP program can be invoked from a user program. It is written as a command processor (CP) for execution under the TSO/E operating system. For more information, see "The sample program, DFH$CUS1" on page 330.

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.

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 100 on page 325) 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 100 on page 325.

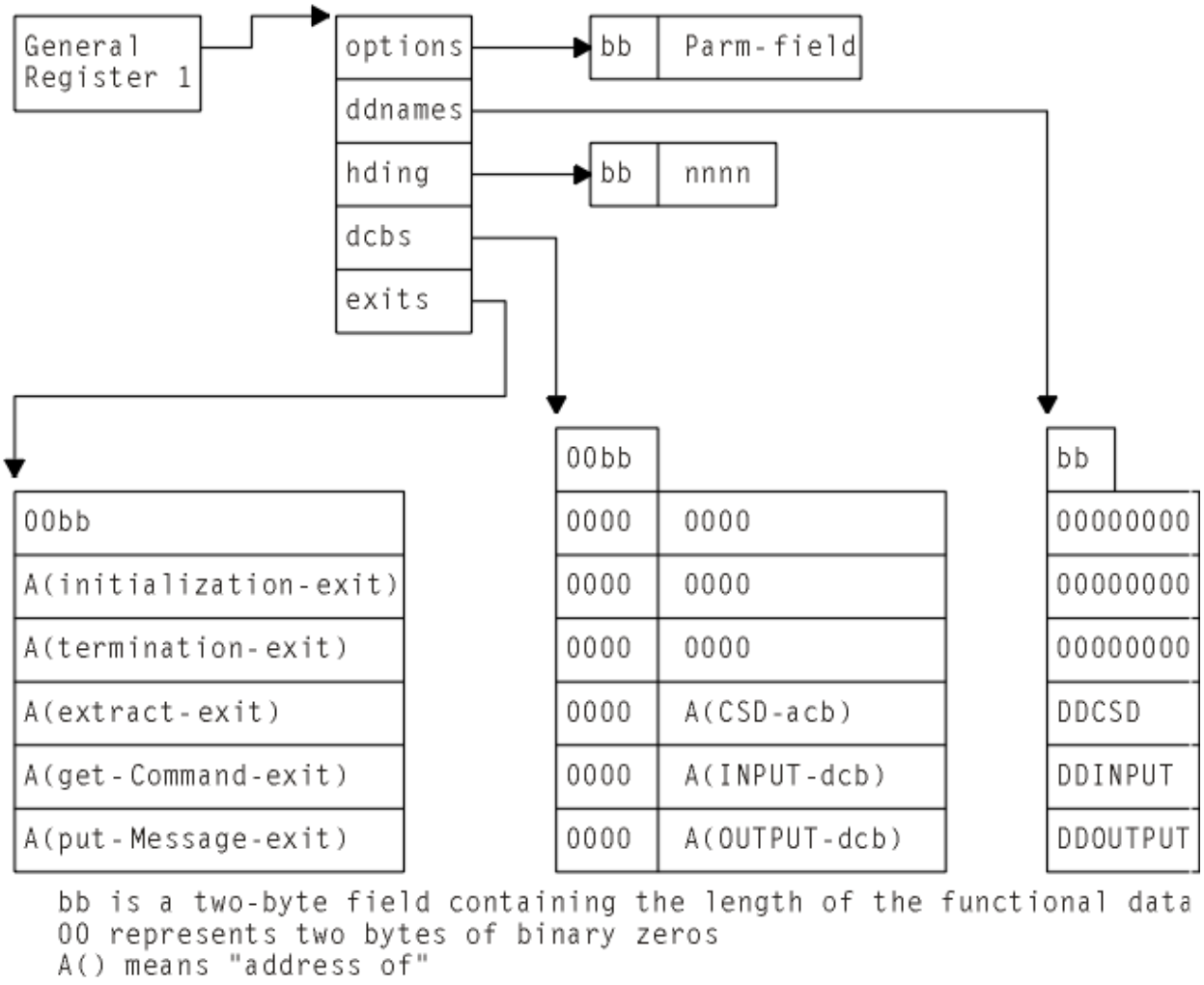


Figure 100. Entry parameters for DFHCSDUP

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 100 on page 325. (The user exits are described in "The user exit points in DFHCSDUP" on page 326.)

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 315.

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 316.) 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 326.

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 323.) 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 101 on page 330 is an example REXX EXEC that invokes DFH$CUS1.

```
/*REXX*/
"ALLOCATE DSN('XXXXX.CICS720.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 101. A REXX program that invokes the DFH$CUS1 sample program

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 CICSTS55.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 application programs.

Example

The job stream in Figure 102 on page 331 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='CICSTS55.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 102. 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 103 on page 331.

Figure 103. 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. Customizing security processing

In CICS Transaction Server for z/OS, Version 5 Release 5 , the only form of security CICS supports is that provided by an external security manager (ESM), such as RACF. CICS uses, by means of the RACROUTE macro, the MVS system authorization facility (SAF) interface to route authorization requests to the external security manager. SAF uses the MVS router as a common system interface for all products providing and requesting resource control. CICS also uses a number of RACF callable services. This section describes how you can customize security processing using RACF exit programs.

To customize security at the SAF level, you can write an MVS Router Exit (ICHRTX00) to customize RACROUTE calls, and an RACF Callable Services Installation Exit (IRRSXT00) to customize RACF callable services.

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 z/OS Security Server RACF Messages and Codes.

For more information about passing control to a user-supplied ESM, see Exit Routine Processing in z/OS MVS Installation Exits.

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 28 on page 333 shows how some formats of the ESM parameter list can be mapped using MVS macros.

Table 28. 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'

Table 28. Mapping the ESM parameter list (continued)		
RACROUTE REQUEST type	**Parameter list mapping macro**	**INSTLN field name**
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 29 on page 334.

Table 29. 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.
3. As a result of RACF APAR OA43999, passwords will no longer be available to the ICHRIX01 user exit when the passwords are valid. In normal usage, the exit will only have access to the password if the password was invalid. This is because the verification and changing of passwords is now performed separately from the sign-on. This has changed the RACF calls made during the sign-on, as well as the data available to user exits invoked as part of those calls. The following steps are performed:
   - RACF service IRRSPW00 is called to verify the supplied password. This service does not drive any user exits. If the password verification fails, or the supplied password is a PassTicket, or the password is valid but there was a previous failure, then a RACROUTE REQUEST=VERIFYX call is made. The ICHRIX01 user exit is invoked and is passed installation data.
   - If a password change operation is requested, a RACROUTE REQUEST=VERIFYX call is made to verify the original password and to perform the password change operation. The ICHRIX01 user exit is invoked and is passed installation data.

- The sign-on uses RACROUTE REQUEST=VERIFY. This call invokes the ICHRIX01 user exit and passes installation data. The password and any new password are not available.

4. There is no RACF user exit for REQUEST=EXTRACT, and no installation parameter data is passed. Any customization must be done using the MVS router exit, ICHRTX00.

For full descriptions of the RACF exit parameter list, see z/OS Security Server RACF Security Administrator's Guide. For more information about CICS security processing using RACF, see RACF facilities.

Mapping the installation data parameter list

The installation data parameter list gives your ESM exit programs access to the CICS security event being processed and details of the current CICS environment. You can map the installation parameter list using the macro DFHXSUXP.

See The installation data parameter list.

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 timeout 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.

PASSWORD_CHANGE (X'90')
Change of password

PASSWORD_VERIFICATION (X'91')
Verification of password

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.

CICS security control points

CICS uses RACROUTE macros and RACF callable services to call the external security manager (ESM). Theses calls are issued at a number of control points. Some calls might not always be issued, because CICS reuses entries for eligible user IDs that have already signed on in the CICS region.

RACROUTE macros

RACROUTE
This macro is the "front end" to the macros described below. The macro calls the MVS router.

RACROUTE REQUEST=VERIFY
This macro is issued at operator sign-on, with the parameter ENVIR=CREATE, and at sign-off, with the parameter ENVIR=DELETE. This macro creates or destroys an ACEE (access control environment element). This macro is issued, with the parameter ENVIR=VERIFY, early in normal sign-on through the **`EXEC CICS SIGNON`** command, but the command is ignored by RACF.

This macro is issued at the following CICS control points.

Each of the following control points relates to ENVIR=CREATE:

- Normal sign-on through **`EXEC CICS SIGNON`**.
- Sign-on of the default user ID DFLTUSER.
- Sign-on of preset-security terminal.
- Sign-on of MRO session.
- Sign-on of LU6.1 session.
- Sign-on of LU6.2 session.
- Sign-on for XRF tracking of any of the above.
- Sign-on associated with the user ID on an attach request, for all operands of ATTACHSEC except LOCAL.
- The first time a userid is authenticated each day.

Each of the following control points relates to ENVIR=DELETE:

- Normal sign-off through **EXEC CICS SIGNOFF**.
- Sign-off when deleting a terminal.
- Sign-off when TIMEOUT expires.
- Sign-off when USRDELAY expires.
- Sign-off of MRO session.
- Sign-off of LU6.1 session.
- Sign-off of LU6.2 session.
- Sign-off for XRF tracking of any of the above.
- Sign-off associated with the user ID on an attach request, for all operands of ATTACHSEC except LOCAL.
- Sign-off because RACF notifies CICS of changes to a user profile, and an attached request associated with that signed-on user ID completes, for all operands of ATTACHSEC except LOCAL.
- Sign-off because RACF notifies CICS of changes to a user profile, and a new attach request is made and the value in the **USRDELAY** system initialization parameter has not expired. This sign-off is followed by a sign-on.
- The first time a userid is authenticated each day.

RACROUTE REQUEST=VERIFYX

This macro creates and deletes an ACEE in a single call. This macro is issued at the following control points:

- When an authentication process that involves password verification is used, and one of the following conditions applies:
  - The password is invalid (following an R_Password or RACROUTE REQUEST=EXTRACT).
  - The previous attempt to log in was invalid.
- Sign-on, as an alternative to VERIFY, when an optimized sign-on is performed for subsequent attach sign-ons across an LU6.2 link with ATTACHSEC(VERIFY) or ATTACHSEC(PERSISTENT).
- Changing a password or password phrase

RACROUTE REQUEST=FASTAUTH

This macro is issued during resource checking, on behalf of a user who is identified by an ACEE. This macro is the high-performance form of REQUEST=AUTH, using in-storage resource profiles, which does not cause auditing to be performed. This macro is issued at the following CICS control points:

- When attaching a local transaction
- When checking link security for transaction attach
- Transaction validation for an MRO task
- CICS resource checking
- Link security check for a CICS resource
- Transaction validation for EDF
- Transaction validation for the transaction being tested (by EDF)
- DBCTL PSB scheduling resource security check
- DBCTL PSB scheduling link security check
- Remote DL/I PSB scheduling resource check
- When checking a surrogate user authority
- QUERY SECURITY with the RESTYPE option

RACROUTE REQUEST=AUTH

This macro provides a form of resource checking with a larger pathlength and causes auditing to be performed. This macro is used as follows:

- After a call to FASTAUTH indicates an access failure that requires logging.

- When a QUERY SECURITY request with the RESCLASS option is used. This option indicates a request for a resource for which CICS has not built in-storage profiles.

RACROUTE REQUEST=LIST
This macro is issued to create and delete the in-storage profile lists needed by REQUEST=FASTAUTH. One REQUEST=LIST macro is required for each resource class. This macro is issued at the following CICS control points:

- When CICS security is being initialized
- When an EXEC CICS PERFORM SECURITY REBUILD command is issued
- When XRF tracks either of these events

RACROUTE REQUEST=EXTRACT
This macro is used in place of R_Password if R_Password is not available (see note 1).

The RACROUTE REQUEST=EXTRACT macro is also issued with the SEGMENT=CICS,CLASS=USER parameters and with the SEGMENT=BASE,CLASS=USER parameters to obtain the national language and user name, at all of the following control points:

- Normal sign-on through EXEC CICS SIGNON
- Sign-on of the default user ID DFLTUSER
- Sign-on of preset security terminal
- Sign-on of MRO session
- Sign-on of LU6.1 session
- Sign-on of LU6.2 session
- Sign-on for XRF tracking of any of those previously mentioned.
- Sign-on associated with the user ID on an attach request, for all operands of ATTACHSEC except LOCAL

The macro is also issued, with the SEGMENT=SESSION,CLASS=APPCLU parameters, during verification of LU6.2 bind security, at the CICS control point for bind of an LU6.2 sessions.

The macro can be used to verify the password of the user when an entry in the user table is reused within the USRDELAY period.

The REQUEST=EXTRACT parameter has no associated RACF user exit, and no installation parameter data is passed. You use the MVS router exit, ICHRTX00, for customization.

For a detailed description of all these macros, see the z/OS Security Server RACROUTE Macro Reference.

z/OS Security Services RACF Callable Services

CICS uses the following callable interfaces for different purposes when calling ESM.

deleteUSP (IRRSDU00): Delete USP
Used for HFS file security.

initACEE (IRRSIA00): Initialize ACEE
Used to obtain userids from a certificate.

initUSP (IRRSIU00): Initialize USP
Used for HFS file security.

R_admin (IRRSEQ00): RACF administration API
Used to validate a certificate label.

R_cacheserv (IRRSCH00): Cache services
Used to obtain or delete an ICRX associated with an ACEE.

R_datalib (IRRSDL00): OCSF data library
Used to extract the information of certificates from the CICS key ring.

R_dcekey (IRRSDK00): Retrieve or set a non-RACF password
Used in LDAP processing.

R_GenSec (IRRSGS00): Generic security API interface
Used for Kerberos support. CICS provides Kerberos support through the **VERIFY TOKEN** and **SIGNON TOKEN** API commands, and through web services configuration.

R_kerbinfo (IRRSMK00): Retrieve or set security server network authentication
Used for Kerberos support to obtain the principle name of a region.

R_ticketserv (IRRSPK00): Parse or extract
Used for Kerberos support. CICS provides Kerberos support through the **VERIFY TOKEN** and **SIGNON TOKEN** API commands, and through web services configuration.

R_usermap (IRRSIM00): Map application user
Used to obtain a user associated with a Kerberos token in Kerberos verification. CICS provides Kerberos support through the **VERIFY TOKEN** and **SIGNON TOKEN** API commands, and through web services configuration.

R_Password (IRRSPW00): Evaluate or encrypt a clear-text password or password phrase
Used for **VERIFY PASSWORD** and **VERIFY PHRASE** API commands, and for the **SIGNON** API command with PASSWORD or PHRASE specified (see note 1).

For a detailed description of these calls, see z/OS Security Server RACF Callable Services.

Note:

1. Requires z/OS 2.2, or z/OS 2.1 with the PTF for APAR CA43999 applied.

Suppressing attach checks for non-terminal transactions

CICS always performs a transaction-attach security check for each transaction attach, even when the transaction has no associated terminal. However, you can bypass transaction-attach security checks for non-terminal transactions while continuing to keep full transaction-attach security for terminal-attached transactions.

CICS always performs the transaction-attach resource check using RACROUTE REQUEST=FASTAUTH, so you need only to provide an ICHRFX01 user exit. The ICHRFX01 routine must issue a zero return code to indicate that the resource check processing is to continue, or a return code of 8 to indicate that the check is to be regarded as successful.

So that the ICHRFX01 exit can determine the circumstances under which it is called, specify the ESMEXITS=INSTLN system initialization parameter for the CICS regions for which you want to control transaction-attach security. Then your ICHRFX01 routine should do the following:

1. Obtain the address of the CICS installation data parameter list, as described in How ESM exit programs access CICS-related information. If this address is zero, either the caller of the RACROUTE macro is not CICS, or it is a CICS region whose behavior you do not want to modify; so exit with a return code of zero.
2. Use the DFHXSUXP macro to map the fields in the installation data parameter list.
3. Confirm that the installation data was created by CICS, by checking that UXPDFHXS is equal to 'DFHXS'. If it is not, exit with a return code of zero.
4. Examine field UXPPHASE in the installation data. If it is not equal to USER_ATTACH_CHECK (X'40'), this is not a transaction attach, so exit with a return code of zero.
5. Examine field UXPTERM in the installation data. If it is nonzero, this is a terminal-related transaction attach, so exit with a return code of zero.
6. If UXPPHASE is USER_ATTACH_CHECK and UXPTERM is zero, then a non-terminal transaction is being attached. Exit with a return code of 8 to indicate to RACF that this check is successful. The function RACROUTE REQUEST=FASTAUTH then completes with a return code of zero, and CICS continues with the attach of the non-terminal transaction.

Global user exits in signon and signoff

CICS provides the XSNON global user exit in EXEC CICS SIGNON processing and the XSNOFF global user exit in EXEC CICS SIGNOFF processing. These exits do not allow you to affect the result of the sign-on or sign-off, but notify you when the user ID associated with a terminal changes.

The exits are further described in Global user exit programs.

Appendix A. 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.

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 31 on page 350. 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 352 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 30. 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'`	

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'`	

Table 30. 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

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'`	

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'`	

Table 30. 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

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

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

Table 30. 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

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.

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 31 on page 350 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 Autoinstalling z/OS Communications Server terminals.

Table 31. TYPETERM device types, with cross-references to z/OS Communications Server logmode entries

TYPETERM device type	**Reference number in Table 30 on page 346**
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

Table 31. TYPETERM device types, with cross-references to z/OS Communications Server logmode entries (continued)

TYPETERM device type	**Reference number in Table 30 on page 346**
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
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

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 32 on page 352, 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 33 on page 352.

Table 32. 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 33. 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 33. 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'00000000000185000007E00' 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'01000001000000000000000'
```

```
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'00800000000185000007E00'
```

```
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'00000000000185000007E00'
```

```
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'00000000000185020507F00'
```

```
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'02800000000185000007E00'
```

```
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 B. 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 Writing a node error program.

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 34 on page 365 shows the message issued and action flags set by DFHZNAC for each terminal error code.

The figures in the “**Action flags set**” column are translated into bit settings and explained in Table 37 on page 380.

Table 34. 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
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

Table 34. Messages issued and flags set by DFHZNAC for specific error codes (continued)

Error code	Symbolic label	Message	Action flags set
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 371	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 371	13
X'48'	TCZOPSIN	DFHZC3461	7 , 8
X'49'	TCZCLSIN	DFHZC3462	7

Table 34. Messages issued and flags set by DFHZNAC for specific error codes (continued)

Error code	Symbolic label	Message	Action flags set
X'4A'	TCZOPACB	DFHZC3463	None
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 371	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

Table 34. Messages issued and flags set by DFHZNAC for specific error codes (continued)

Error code	Symbolic label	Message	Action flags set
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
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

Table 34. Messages issued and flags set by DFHZNAC for specific error codes (continued)

Error code	Symbolic label	Message	Action flags set
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
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

Table 34. Messages issued and flags set by DFHZNAC for specific error codes (continued)

Error code	Symbolic label	Message	Action flags set
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
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

Table 34. Messages issued and flags set by DFHZNAC for specific error codes (continued)			
Error code	**Symbolic label**	**Message**	**Action flags set**
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'	TCZRTHS	DFHZC3403	2 3 7 9 10 11 23 24
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 35. CICS messages associated with z/OS Communications Server errors			
Message	**Symbolic label**	**Error code**	**Action flags set**
DFHZC0101	TCZPRDTO	X'5D'	None

Table 35. CICS messages associated with z/OS Communications Server errors (continued)

Message	Symbolic label	Error code	Action flags set
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
DFHZC2414	TCZSRCSP	X'80'	None
DFHZC2416	TCZSRCVH	X'13'	7 18 24

Table 35. CICS messages associated with z/OS Communications Server errors (continued)			
Message	**Symbolic label**	**Error code**	**Action flags set**
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

Table 35. CICS messages associated with z/OS Communications Server errors (continued)

Message	Symbolic label	Error code	Action flags set
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
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
DFHZC3403	TCZRTHS	X'ED'	2 3 7 9 10 11 23 24
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

Table 35. CICS messages associated with z/OS Communications Server errors (continued)			
Message	**Symbolic label**	**Error code**	**Action flags set**
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
DFHZC3431	TCZSTIN	X'EF'	2 3
DFHZC3432	TCZSTLER	X'EA'	2 3
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

Table 35. CICS messages associated with z/OS Communications Server errors (continued)

Message	Symbolic label	Error code	Action flags set
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
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

Table 35. CICS messages associated with z/OS Communications Server errors (continued)			
Message	**Symbolic label**	**Error code**	**Action flags set**
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
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

Table 35. CICS messages associated with z/OS Communications Server errors (continued)			
Message	**Symbolic label**	**Error code**	**Action flags set**
DFHZC6598	TCZXRFVS	X'56'	None

DFHZNAC: default actions for system sense codes

Table 36 on page 378 shows the message issued and action flags set by DFHZNAC for each inbound system sense code received. The figures in the "**Action flags set**" column are translated into bit settings and explained in Table 37 on page 380.

Table 36. 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
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

Table 36. Messages issued and flags set by DFHZNAC for specific sense codes (continued)		
Sense code	**Message**	**Action flags set**
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 37 on page 380 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 34 on page 365 and Table 36 on page 378.

Table 37. 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 37. 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

Notices

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

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property rights may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

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

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

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

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

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

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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

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

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 Client Relationship Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

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

Programming interface information

IBM CICS supplies some documentation that can be considered to be Programming Interfaces, and some documentation that cannot be considered to be a Programming Interface.

Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 5 (CICS TS 5.5) are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- CICS security
- Developing for external interfaces
- Reference: application development
- Reference: system programming
- Reference: connectivity

Information that is NOT intended to be used as a Programming Interface of CICS TS 5.5, but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- Troubleshooting and support
- Reference: diagnostics

If you access the CICS documentation in manuals in PDF format, Programming Interfaces that allow the customer to write programs to obtain the services of CICS TS 5.5 are included in the following manuals:

- Application Programming Guide and Application Programming Reference
- Business Transaction Services

- Customization Guide
- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- Installation Guide
- Security Guide
- CICS Transactions
- CICSPlex System Manager (CICSPlex SM) Managing Workloads
- CICSPlex SM Managing Resource Usage
- CICSPlex SM Application Programming Guide and Application Programming Reference
- Java Applications in CICS

If you access the CICS documentation in manuals in PDF format, information that is NOT intended to be used as a Programming Interface of CICS TS 5.5, but that might be misconstrued as Programming Interfaces, is included in the following manuals:

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSPlex SM Problem Determination Guide

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.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Apache, Apache Axis2, Apache Maven, Apache Ivy, the Apache Software Foundation (ASF) logo, and the ASF feather logo are trademarks of Apache Software Foundation.

Gradle and the Gradlephant logo are registered trademark of Gradle, Inc. and its subsidiaries in the United States and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

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

Red Hat®, and Hibernate® are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Spring Boot is a trademark of Pivotal Software, Inc. in the United States and other countries.

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

Zowe™, the Zowe logo and the Open Mainframe Project™ are trademarks of The Linux Foundation.

The Stack Exchange name and logos are trademarks of Stack Exchange Inc.

Terms and conditions for product documentation

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

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

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

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

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

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

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

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

IBM online privacy statement

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 (PII) is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect PII. If this Software Offering uses cookies to collect PII, specific information about this offering's use of cookies is set forth below:

For the CICSPlex SM Web User Interface (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 PII for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface (data interface):
Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other PII for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface ("hello world" page):
Depending upon the configurations deployed, this Software Offering may use session cookies that do not collect PII. 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 PII 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 Privacy Policy and IBM Online Privacy Statement, the section entitled *Cookies, Web Beacons and Other Technologies* and the IBM Software Products and Software-as-a-Service Privacy Statement.

Index

Special Characters

Numerics

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Z