z/OS

MVS Installation Exits

Version 2 Release 1

SA23-1381-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 403.

This edition applies to Version 2 Release 1 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

About this document

This document supports z/OS® (5650-ZOS). This document identifies and describes the installation exits that you can use to modify your MVS™ system.

Who should use this document

This document is intended for a systems programmer who wants to modify the processing of an MVS system. The document assumes that the reader can code in assembler language and read assembler, loader, and linkage editor output.

Although the documentation for the exits in this document follows a standard format, each exit has unique features and requirements that you need to know about. In general, the introductory section to each exit describes the exit's functions and the possible uses for the exit. The remaining sections point out specific considerations for coding the exit.

How to use this document

Because all of the exits perform the general function of modifying the existing code, you may find it helpful to read Section 1 to develop a general idea of what an exit is and how it functions as a part of the MVS system.

Once you have determined the type of modification you want to make to your system, you might follow these general steps when using the document:

- Use the table of contents to locate the exit that might fit your needs.
- Turn to the description of the exit and use the introductory section, which provides a description of the exit routine and its possible uses, to determine if the exit routine can meet your needs.
- Use the detailed information to design, code, and install your installation exit routine.

How each exit is organized

The exit routines in this document follow a format to allow you to locate information about the exit routine you are coding. The list that follows describes how each exit can be organized. However, some of these topics may not apply to every exit, such as I/O restrictions. A topic that does not apply to the exit is not discussed for that exit.

- Exit routine name and description.
- Introductory material, which describes the function of the exit routine and lists possible uses for the routine.
- Installing the exit.
- Exit routine environment, which contains specific environmental considerations that you need to know, such as the address space that the exit runs in, the AMODE and RMODE for the exit, and locks or ENQs that the exit routine might hold when it gets control.
- Exit recovery environment, which describes the type of recovery environment for the exit routine.
- Processing.
- Programming considerations.

- Macro instructions and restrictions, which contain specific macro requirements or restrictions that you must keep in mind when coding the exit routine.
- I/O restrictions.
- Entry specifications, which includes a list of the register contents when the exit routine receives control and, when applicable, describes any related parameter lists for the exit routine.
- Return specifications, which lists the register contents expected when the exit routine returns control to the calling program.
- Coded example of the exit.

Where to find more information

Where necessary, this document references information in other documents, using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US
4. Fax the comments to us, as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
  z/OS V2R1.0 MVS Installation Exits
  SA23-1381-00
- The topic and page number that is related to your comment.
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS support page (http://www.ibm.com/systems/z/support/).

z/OS Version 2 Release 1 summary of changes

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS Introduction and Release Guide*

Part 1. Introduction

In certain areas of processing, your installation might need to customize its MVS system to an extent not available through standard options, such as initialization parameters and operator commands. IBM® provides installation exit points for this purpose.

There are installation exit points located throughout the MVS system code. When an exit is invoked, the system passes control to an IBM-supplied routine that might or might not perform actual processing. When you provide an exit routine, control passes to that routine, which does its special processing. At the end of the user routine, control returns to a specified point in the system code.

Chapter 1. All About Exit Routines

The following section contain introductory information about installation exit routines. To start, it is helpful to review the following terminology.

- **Installation Exit Point**. An installation exit point is a specific point in component processing at which control passes to an installation exit routine. The macro that calls the user routine uses parameter values from keywords supplied at the initialization of the component. In most cases, the initialization statement defaults are set to cause the exit point to be ignored. However, you can modify macro processing by specifying parameters supplied with operator commands.
- **Replaceable Module**. A replaceable module is a load module that the user installation can update, alter, or completely replace. Some replaceable modules are IBM-supplied code that performs a specific function, while others simply branch directly back to the calling program without any additional processing. When you write your exit routine, you replace the IBM-supplied routine with your own. Both the name of the routine and its library location are predefined.
- **Installation Exit Name List**. An installation exit name list is similar to an installation exit point. However, rather than directly calling the installation exit routine, the macro addresses a CSECT that contains a list of installation exit routine names. The installation exit routines are then invoked sequentially as they appear in the name list. This processing ends in one of two ways:
  - One of the installation exit routines issues a terminating return code.
  - The end of the name list is reached.

  Your installation provides the routine names by updating, reassembling, and again linkediting the name list CSECT. (It is possible to update the CSECT temporarily using SPZAP. See *z/OS MVS Diagnosis: Tools and Service Aids* for more information on SPZAP.)

  If you provide no exit routines, the name list contains blank entries.

  In both the installation exit point and installation exit name list methods, the installation defines the installation exit routine names.
- **Dynamic Exit**. A user-written or system exit that has been defined to the dynamic exits facility. See "Dynamic Exits Facility" on page 5 for more information.

Assembling Installation Exit Routines

To ensure all necessary macros are resolved during assembly of your exit routines, use the same SYSLIB concatenation as your SMP procedure or SYSLIB DDDEF statement.

Link editing an Installation Exit Routine into a Library

For information about link editing a dynamic exit routine, see "Link editing a Dynamic Exit Routine into a Library" on page 6.

The example in Figure 1 on page 4 shows how to make an installation exit routine available to the system by link editing it into a system library.

```
//LKUSRPGM     JOB     MSGLEVEL=(1,1)
//            EXEC    PGM=IEWL,PARM='XREF,LET,LIST,NCAL'
//SYSPRINT     DD      SYSOUT=A
//SYSUT1       DD      UNIT=SYSDA,SPACE=(TRK,10)
//SYSLMOD      DD      DSNAME=SYS1.LINKLIB,DISP=SHR
//SYSLIN       DD      *
    object deck
         NAME       EXITNAME(R)
/*
```

Figure 1. Example: Link editing an installation exit routine

EXEC Statement: invokes the linkage editor and requests maximum diagnostic listing. For a description of module attributes that you can assign to your routine, see *z/OS MVS Program Management: Advanced Facilities*.

SYSPRINT DD Statement: defines the message data set.

SYSLMOD DD Statement: defines the output data set, in this case the link library, SYS1.LINKLIB. The output data set can also be a permanent library to be referenced by a JOBLIB or STEPLIB DD statement; in that case, the SYSLMOD DD statement could be coded as follows:

```
//SYSLMOD  DD   DSNAME=MYLIB,UNIT=3380,VOL=SER=666666,
//              DISP=(NEW,KEEP),SPACE=(1024,(20,2,1))
```

SYSLIN DD Statement: defines the input data set, in this example the object code for the user program.

NAME Control Statement: specifies the member name, and thus the program name, to be assigned to the user program. In this example, the member name is EXITNAME.

Programming Considerations for Installation Exit Routines

Each MVS component has individual programming conventions. Components that have similar functions have similar requirements. Most requirements, however, are unique. Because of these differences among component requirements, we cannot provide universal rules for coding installation exit routines. The specific guidelines we can give are:

- Whenever a macro is provided to perform a service, use the macro.
- Upon entry to your exit routine, save all register contents and restore them before returning to your calling routine. An exception to this guideline is the use of return and reason codes. In many cases, you must insert a return code in register 15, and reason codes in registers 0 and 1 before returning to the calling program.
- Under no circumstances should you assume an interface (such as contents of a register) that is not specifically documented.
- **Exits That Use a z/OS UNIX Callable Service**: Any exit that uses a z/OS UNIX callable service needs to be written based on the current environment. If the task is not dubbed on entry to the exit (STCBOTCB=0), the exit must undub (perform a CALL BPX1MPC) at completion if the exit has called any other z/OS UNIX callable service callable service. If the task is already dubbed (STCBOTCB does not equal 0), the exit should leave the environment as it was on entry, which would require closing any files that it opens and leaving signal states unchanged.

Dynamic Exits Facility

The dynamic exits facility is a set of services that you can use through any of the following methods:

- The EXIT statement of the PROGxx parmlib member. The EXIT statement allows an installation to add exit routines to an exit, change the state of an exit routine, delete an exit routine for an exit, undefine an implicitly defined exit, and change the attributes of an exit.

  The PROGxx EXIT statement interacts with the PROG=xx parameter of IEASYSxx and the SET PROG=xx command. At IPL, operators can use PROG=xx to specify the particular PROGxx parmlib member the system is to use. During normal processing, operators can use the SET PROG=xx command to set a current PROGxx parmlib member. See *z/OS MVS Initialization and Tuning Reference* for information about the PROGxx parmlib member.
- The SETPROG EXIT operator command. This command performs the same functions as the EXIT statement of the PROGxx parmlib member. See *z/OS MVS System Commands* for information about the SETPROG EXIT command.
- The CSVDYNEX macro. The CSVDYNEX macro can be used to define exits to the dynamic exits facility, control their use within a program, and associate one or more exit routines with those exits. It can also be used to associate exit routines with the existing SMF and allocation exits, which have been defined to the dynamic exits facility. The CSVDYNEX macro provides a superset of the functions available through the SETPROG EXIT operator command and the EXIT statement of the PROGxx parmlib member. See *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* for information about the CSVDYNEX macro.

An installation can use any of these methods to control dynamic exits. For example, an exit routine can be associated with an exit using the CSVDYNEX ADD request, the SETPROG EXIT,ADD operator command, or the EXIT statement of PROGxx.

The following exits have been defined to the dynamic exits facility:

- Allocation exits:
  - IEF_ALLC_OFFLN— Allocated/Offline Device Installation Exit
  - IEFDB401— Allocation Input Validation Routine
  - IEF_SPEC_WAIT— Specific Waits Installation Exit
  - IEF_VOLUME_ENQ— Volume ENQ Installation Exit
  - IEF_VOLUME_MNT— Volume Mount Installation Exit
- Automatic restart management exits:
  - IXC_ELEM_RESTART— Element Restart Exit
  - IXC_WORK_RESTART— Workload Restart Exit
- SMF exits:
  - IEFACTRT— SMF Job/Job Step Termination Exit
  - IEFUAV— User Account Validation Exit
  - IEFUJI— Job Initiation Exit
  - IEFUJP— Job Purge Exit
  - IEFUJV— Job Validation Exit
  - IEFUSI— Step Initiation Exit
  - IEFUSO— SYSOUT Limit Exit

  - IEFUTL— Time Limit Exit
  - IEFU29— SMF Dump Exit
  - IEFU83— SMF Record Exit
  - IEFU84— SMF Record Exit
  - IEFU85— SMF Record Exit
- Subsystem interface (SSI) exit:
  - IEFJFRQ— Subsystem Function Request Exit
- SVC dump (SDUMP) exits:
  - IEASDUMP.QUERY
  - IEASDUMP.GLOBAL
  - IEASDUMP.LOCAL
  - IEASDUMP.SERVER

Link editing a Dynamic Exit Routine into a Library

You can link edit a dynamic exit routine into a library by:

- Link editing it into a data set that is made part of the PLPA, MLPA, or FLPA at IPL-time
- Link editing it into a data set that is part of the LNKLST concatenation
- Link editing it into IEANUC0x
- Link editing it into any PDS/PDSE and naming that data set using the DSNAME option of:
  - The SETPROG EXIT command
  - The EXIT ADD statement of a PROGxx parmlib member
  - The ADD request of the CSVDYNEX macro.

Replacing a Dynamic Exit Routine

There are two ways to replace a dynamic exit routine:

1. You can delete the current exit routine and add a replacement routine that uses either the same or another name, or
2. You can change the state of the current routine to "inactive" and then later change it back to "active."

You can use any of the following methods to accomplish the replacement:

- Use the CSVDYNEX macro.
- Use the SETPROG EXIT command.
- Use the SET PROG= command with new or modified EXIT statements in a PROGxx parmlib member.
- Use the PROG= system parameter with new or modified EXIT statements in a PROGxx parmlib member when you re-IPL.

You will make the most efficient use of storage if an exit routine that is associated with more than one exit resides in the LPA. However, it is easier to replace a dynamic exit routine that does not reside in the LPA.

If you are replacing an exit routine that *is* in the LPA, observe these two cautions so that the system does not find and use the original (unchanged) copy of the module in the LPA:

1. Use the DSNAME parameter of the ADD statement to specify the library into which you have placed the modified load module.

2. Do not use the method that changes the state of the exit routine to inactive and then back to active, unless you use the dynamic LPA function to add or replace the modified exit module to the system.

Providing Security for Dynamic Exits

An installation can control access to programs that create and call dynamic exits by authorizing specific users or groups of users to issue the various CSVDYNEX requests.

The exit services tables contain lists of exits with associated exit routines. To protect the exit services tables, set up RACF® FACILITY resource class profiles that protect the following entities:

- CSVDYNEX.exitname.DEFINE
- CSVDYNEX.exitname.modname
- CSVDYNEX.exitname.UNDEFINE
- CSVDYNEX.exitname.ATTRIB
- CSVDYNEX.LIST
- CSVDYNEX.exitname.CALL
- CSVDYNEX.exitname.RECOVER
- Grant UPDATE authority to CSVDYNEX.exitname.DEFINE to users who are authorized to define dynamic exits (CSVDYNEX DEFINE request).
- Grant UPDATE authority to CSVDYNEX.exitname.modname to users who are authorized to add, delete, or modify exit routines for dynamic exits (CSVDYNEX ADD, REPLACE, MODIFY, and DELETE requests).
- Grant UPDATE authority to CSVDYNEX.exitname.UNDEFINE to users who are authorized to remove the definition of a dynamic exit (CSVDYNEX UNDEFINE request).
- Grant UPDATE authority to CSVDYNEX.exitname.ATTRIB to users who are authorized to change the attributes of dynamic exits (CSVDYNEX ATTRIB request).
- Grant READ authority to CSVDYNEX.LIST to users who are authorized to list information about dynamic exits (CSVDYNEX LIST request).
- Grant READ authority to CSVDYNEX.exitname.CALL to users who are authorized to call exit routines for dynamic exits (CSVDYNEX CALL FASTPATH=NO request).
- Grant READ authority to CSVDYNEX.exitname.RECOVER to users who are authorized to provide recovery for exit routines associated with dynamic exits (CSVDYNEX RECOVER request).

The SET PROG= and SETPROG commands are restricted to consoles with SYS authority. The DISPLAY PROG command can be issued on any console with INFO or higher authority You can prevent operators from issuing particular requests by setting up the RACF profiles listed above for the ADD, MODIFY, DELETE, and ATTRIB requests.

Part 2. The Exits

The installation exits described in this section of the document are functions supported in the MVS base control program (BCP) code. The remaining installation exit functions found in various MVS components and related products are listed in Part 3, "Installation Exit Directory," on page 335.

Chapter 2. ASREXIT — SYMREC Authorization Exit

Topics for This Exit Appear as Follows:

To allow unauthorized programs to write symptom records to the logrec data set (or to a job log or to both) through the use of the SYMREC macro, the SYMREC authorization exit, ASREXIT, must be in effect.

IBM provides the following sample ASREXIT routines for your installation's use:

- ASREXT0 allows all unauthorized programs to write symptom records to SYS1.LOGREC.
- ASREXT1 allows only unauthorized programs *that reside in APF-authorized libraries* to write symptom records to SYS1.LOGREC.

If one of these IBM-supplied routines serves the needs of your installation, use it instead of coding your own routine (see "Coded Examples of the Exit Routine" on page 14). If you plan to code your own routine, ASREXT0 and ASREXT1 appear in SYS1.SAMPLIB. You can refer to them as examples when coding your routine.

You can use the ASREXIT interface to:

- Allow all or only particular unauthorized programs to write symptom records to SYS1.LOGREC.
- Cause unauthorized programs to write symptom records to a job log instead of, or in addition to, SYS1.LOGREC.

If you do not install the ASREXIT routine, MVS will not allow unauthorized programs to write symptom records to SYS1.LOGREC.

For information on the SYMREC macro, see *z/OS MVS Programming: Assembler Services Guide*.

Installing the Exit Routine

The exit must be linkedited with the name ASREXIT into SYS1.LINKLIB or any library in the LNKLST concatenation. To activate the exit routine, refresh LLA through the F LLA,REFRESH command.

For general instructions on installing an exit routine, see “Link editing an Installation Exit Routine into a Library” on page 3.

Exit Routine Environment

ASREXIT receives control running under the unit of work that invoked the SYMREC service, in the following environment:

- In supervisor state with PSW key 0
- Enabled or disabled for interrupts
- AMODE 31 and RMODE ANY
- Primary=home address space of the unit of work that issues the SYMREC request.
- SRB or task mode. PSATOLD=0 means SRB mode, and nonzero means task mode.

  Note: Whether the exit routine receives control in task or SRB mode is dependent on the unit of work that issues the SYMREC request.

Exit Recovery: ASREXIT should provide its own recovery.

If ASREXIT does not provide a recovery routine, or if an exit routine error percolates beyond the exit's recovery, the system's ESTAEX recovery routine will get control. The ESTAEX will record information in the SDWA and request an SDUMP.

If ASREXIT abends, the system will not allow the symptom record to be written to SYS1.LOGREC. The exit will be invoked for the next unauthorized program that attempts to write symptom records.

Exit Routine Processing

When you install ASREXIT, the system will invoke the exit routine whenever an unauthorized program issues the SYMREC macro to write symptom records to the SYS1.LOGREC data set. The system passes to the exit in register 1 the address of fullword, which contains an address of the SYMREC authorization exit parameter list (mapped by ASREPL). The SYMREC authorization exit parameter list contains the following:

- A program name
- The job step name
- The address of the symptom record
- An indication of whether the program originated from an APF-authorized library.

ASREXIT runs under an RB that issues a SYMREC request. The values of the program name and the APF-authorized library indication in the SYMREC authorization exit parameter list are not necessarily those of the RB that issued the SYMREC request. The system obtains the program name and the APF-authorized library indication from the MAJOR CDE pointed to by the oldest RB on the TCB RB chain. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for information on the CDE data area. Based on the program name and the authorized library indication, ASREXIT indicates (by placing a value in the EPLRETC field of the exit parameter list) whether to reject the request or allow the program to write the symptom record to:

- SYS1.LOGREC
- The job log
- Both SYS1.LOGREC and the job log.

If you code an ASREXIT routine, instead of installing one of the two IBM-supplied routines, you will probably use the routine to allow only ***specific*** unauthorized programs to write symptom records, or to cause unauthorized programs to write symptom records to their job log instead of, or in addition to, SYS1.LOGREC.

Using the Job Log: If you intend to restrict unauthorized programs from writing symptom records to SYS1.LOGREC, but still want to collect symptom records from unauthorized programs for debugging purposes, code the ASREXIT routine to cause unauthorized programs' symptom records to be written to their job log (by issuing WTOs with routing code 11). Application programmers can then view symptom records in the job log directly, rather than having the system programmer search SYS1.LOGREC for the symptom records.

If you do not install an ASREXIT routine, unauthorized programs cannot write symptom records to a job log.

Programming Considerations

Code the ASREXIT exit routine to be reentrant. A new copy of ASREXIT is loaded into storage from SYS1.LINKLIB for every request to write a SYMREC.

Entry Specifications

The system passes the address of the exit parameter list to ASREXIT.

Registers at Entry: The contents of the registers on entry to ASREXIT are as follows.

Register	Contents
0	Not applicable
1	Pointer to the address of the exit parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit

Parameter List Contents: Register 1 contains a pointer to the address of the exit parameter list, the EPL. The EPL contains the program name, job step name, address of the symptom record, and an indication of whether the program resides in an APF-authorized library.

The EPL is mapped by the ASREPL macro (data area ASREPL), which resides in SYS1.MODGEN. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for a mapping of the ASREPL data area.

Return Specifications

ASREXIT indicates to the system whether to cancel the request or allow the symptom record to be written by placing one of the following values in the EPLRETC field of the EPL:

Value	Explanation
X'00'	Write the symptom record to SYS1.LOGREC
X'04'	Write the symptom record to ***both*** SYS1.LOGREC and to the job log
X'08'	Write the symptom record to the job log only
X'0C'	Do not write the symptom record. The system returns to the caller both a system return code (X'0C') and reason code (X'F1C') indicating that request was rejected for either of the following reasons: • The exit routine rejected the request. • The exit routine was not installed.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0,1	Not relevant
2-14	Restored to contents at entry
15	Not relevant

Coded Examples of the Exit Routine

IBM provides two SYMREC authorization exit routines for your installation's use, ASREXT0 and ASREXT1.

Use ASREXT0 to allow ***all*** unauthorized programs to write symptom records to SYS1.LOGREC. This exit routine always returns a value of X'00' (grant the request).

Use ASREXT1 to allow only unauthorized programs (that is, programs that were not linkedited with authorization code AC=1), that you have installed in an APF-authorized library to write symptom records to SYS1.LOGREC. When it is invoked, ASREXT1 checks to see whether the calling program resides in an APF-authorized library (the system sets the EPLAPFL bit to 1). If EPLAPFL is set to 1, ASREXT1 returns a value of X'00' (grant the request). Otherwise, ASREXT1 returns a value of X'0C' (reject the request).

ASREXT0 and ASREXT1 are provided in SYS1.SAMPLIB. To install either exit routine, you must linkedit the routine with the name ASREXIT into SYS1.LINKLIB or any library in the LNKLSTxx concatenation.

When you assemble ASREXT0 and ASREXT1, ensure SYS1.MODGEN is included in Assembler SYSLIB concatenation.

For more information on APF-authorized libraries, see *z/OS MVS Programming: Assembler Services Guide*.

Chapter 3. CNZ_MSGTOSYSLOG — Message To Syslog Exit

Topics for This Exit Appear as Follows:

CNZ_MSGTOSYSLOG receives control from the system when a message is sent to the SYSLOG. Every message line that is sent to syslog will be passed to the exit routines active at the exit point. Multi-line messages will be presented as major line first, then major and each minor (one at a time). For example, a multi-line message with 1 major and 3 minor lines will result in the exit routines receiving control 4 times:

- 1st Time - For the major line
- 2nd Time - For the first minor line
- 3rd Time - For the second minor line
- 4th Time - For the last minor line

Code a CNZ_MSGTOSYSLOG exit routine when you want to view all messages being sent to the SYSLOG.

Controlling the Exit Routine Through the Dynamic Exits Facilities

IBM has defined the CNZ_MSGTOSYSLOG exit to the dynamic exits facility. You can refer to the exit by the name CNZ_MSGTOSYSLOG. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

1. The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error.
2. The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system disables the exit routine after one abend.

Exit Routine Environment

The exit receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31.
- With no locks or ENQs held.
- In the CONSOLE address space.

Exit Recovery: The exit routine should NOT provide its own recovery. If recovery is necessary, set up an EUT functional recovery routine (FRR) instead of an ESTAE-type recovery routine. This will improve system performance by shortening the path length of the exit routine.

If the exit routine abnormally terminates, and the exit routine does not provide its own recovery, or the error percolates beyond the exit's recovery routine, a system recovery routine will get control.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

MVS invokes the CNZ_MSGTOSYSLOG exit routine every time a message is sent to the SYSLOG. If any CNZ_MSGTOSYSLOG routines are specified to the dynamic exits facility, a parameter list (mapped by macro CNZMYM2S) is passed that contains the following information about the message:

- Whether the message is a MLWTO
- Whether the message is for a minor line WQE
- Whether the message is the last line of a MLWTO
- Pointer to the single or major line WQE (Mapped by macro IHAWQE, data area WQE for single WQE and data area WMJM for major line WQE)
- Pointer to the current minor line WQE (or zero) (Mapped by macro IHAWQE, data area WMNM)

  Note: Exactly one minor line is passed to the exit routine.
- Pointer to a 4K workarea that can be used by the exit (Each exit routine will share the same 4K workarea. It is up to the exit routine to initialize the workarea.)

Programming Considerations

Observe the following conventions when coding the CNZ_MSGTOSYSLOG exit routine:

- Code the exit routine to be reentrant.
- Do not code the exit routine to issue messages to the SYSLOG.

Performance Considerations

Message to syslog processing may impact performance; therefore, consider the following recommendations so that system performance is not degraded:

- Exit routines installed at the CNZ_MSGTOSYSLOG exit point should not perform operations that might degrade system performance, such as issuing WAIT requests, issuing requests for large amounts of dynamic storage, or issuing I/O requests. To reduce the need for storage requests, the system provides a 4K workarea that your exit routine can use for dynamic storage. The M2SL_WorkArea@ field in the CNZMYM2S parameter mapping points to the 4K workarea. The exit routine must clear the workarea before using it, because all exit routines installed at the CNZ_MSGTOSYSLOG exit point use the same workarea.
- The exit routine should NOT provide its own recovery. If recovery is necessary, set up an EUT functional recovery routine (FRR) instead of an ESTAE-type recovery routine. This will improve system performance by shortening the path length of the exit routine.

Entry Specifications

The system passes the address of the exit parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of a pointer to the exit parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit

Parameter Descriptions: Register 1 contains the address of a pointer to the exit parameter list, the M2SL, which is mapped by macro CNZMYM2S.

Return Specifications

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-15	Restored to contents at entry

Coded Example of the Exit Routine

The following is an example of an CNZ_MSGTOSYSLOG exit:

```
         TITLE 'M2SLEXIT - SAMPLE MESSAGE TO SYSLOG EXIT'
***START OF SPECIFICATIONS*************************************
*                                                             *
* MODULE NAME         =  M2SLEXIT                             *
*                                                             *
* DESCRIPTIVE NAME    =  SAMPLE MESSAGE TO SYSLOG EXIT.       *
*                                                             *
* FUNCTION            =  THIS EXIT WILL DEMONSTRATE HOW TO    *
*                        PROCESS THE DATA IN THE PARAMETER    *
*                        LIST.                                *
*                                                             *
```

```
*   OPERATION         =  IDENTIFIES THE MESSAGE LINE PASSED IN *
*                        THE PARAMETER LIST AND BRANCHES TO    *
*                        THE APPROPRIATE SECTION TO PERFORM    *
*                        PROCESSING BASED ON THE MESSAGE LINE. *
*                                                              *
* ENTRY POINT         =  M2SLEXIT                              *
*                                                              *
*   PURPOSE           =  DEMONSTRATE HOW TO USE THIS EXIT.     *
*                                                              *
*   LINKAGE           =  BALR                                  *
*                                                              *
*   INPUT DATA        =  REG1  ADDRESS OF THE PARAMETER LIST   *
*                        REG13 ADDRESS OF STANDARD SAVE AREA   *
*                        REG14 RETURN ADDRESS                  *
*                        REG15 ENTRY POINT ADDRESS             *
*                                                              *
*   REGISTERS SAVED   =  REG0 - REG15                          *
*                                                              *
*   REGISTER USAGE    =  REG0  - USED FOR BASING               *
*                        REG1  - PARAMETER REGISTER            *
*                        REG2  - WORK REGISTER                 *
*                        REG3  - WORK REGISTER                 *
*                        REG4  - WORK REGISTER                 *
*                        REG5  - WORK REGISTER                 *
*                        REG6  - POINTER TO PARAMETER LIST     *
*                        REG7  - NOT USED                      *
*                        REG8  - NOT USED                      *
*                        REG9  - NOT USED                      *
*                        REG10 - NOT USED                      *
*                        REG11 - POINTER TO 4K WORK AREA       *
*                        REG12 - MODULE BASE REGISTER          *
*                        REG13 - POINTER TO STANDARD SAVE AREA *
*                        REG14 - RETURN POINT                  *
*                        REG15 - NOT USED                      *
*                                                              *
*   REGISTERS RESTORED = REG0 - REG14                          *
*                                                              *
*   CONTROL BLOCKS    =                                        *
*     NAME     MAPPING MACRO   REASON USED           USAGE     *
*     ----     -------------   -----------           -------   *
*     M2SL     CNZMYM2S        EXIT PARAMETER LIST    R        *
*     WQE      IHAWQE          WTO QUEUE ELEMENT      R        *
*                                                              *
*   KEY = R-READ, W-WRITE, C-CREATE, D-DELETE                  *
*                                                              *
*   TABLES            =  NONE                                  *
*                                                              *
*   MACROS            =  NONE                                  *
*                                                              *
*   MESSAGES          =  NONE                                  *
*                                                              *
*   MODULE TYPE       =  CSECT                                 *
*                                                              *
*     ATTRIBUTES      =  REENTRANT, REUSABLE, AMODE 31,        *
*                        RMODE ANY                             *
*                                                              *
****************************************************************
EJECT
M2SLEXIT CSECT
M2SLEXIT AMODE 31                    31-BIT ADDRESSING MODE
M2SLEXIT RMODE ANY                   31-BIT RESIDENCE
SPACE 1
****************************************************************
*                                                              *
*         REGISTER ASSIGNMENTS                                 *
*                                                              *
****************************************************************
```

```
REG0     EQU   0                       REGISTER 0
REG1     EQU   1                       REGISTER 1
REG2     EQU   2                       REGISTER 2
REG3     EQU   3                       REGISTER 3
REG4     EQU   4                       REGISTER 4
REG5     EQU   5                       REGISTER 5
M2SLPTR  EQU   6                       REGISTER 6 - PTR TO M2SL
REG6     EQU   6                       REGISTER 6
REG7     EQU   7                       REGISTER 7
REG8     EQU   8                       REGISTER 8
REG9     EQU   9                       REGISTER 9
REG10    EQU   10                      REGISTER 10
REG11    EQU   11                      REGISTER 11
DATAREG  EQU   11                      REGISTER 11 - WORKAREA
REG12    EQU   12                      REGISTER 12
BASEREG  EQU   12                      REGISTER 12 - MOD BASE
REG13    EQU   13                      REGISTER 13
REG14    EQU   14                      REGISTER 14
REG15    EQU   15                      REGISTER 15
*****************************************************************
*                                                               *
*        STANDARD ENTRY LINKAGE                                 *
*                                                               *
*****************************************************************
         SAVE  (14,12)                 SAVE REGISTERS
         BASR  BASEREG,REG0            ESTABLISH MODULE BASE
         USING *,BASEREG               ESTABLISH ADDRESSABILITY
         LR    M2SLPTR,REG1            ESTABLISH ADDRESSABILITY
         USING M2SL,M2SLPTR            TO THE PARAMETER LIST
*****************************************************************
*                                                               *
*        SET ADDRESSABILITY AND INITIALIZE 4K WORKAREA          *
*                                                               *
*****************************************************************
         L     DATAREG,M2SL_WORKAREA@  ESTABLISH ADDRESSABILITY
         USING DATAAREA,DATAREG        TO 4K WORKAREA
         LR    REG2,DATAREG            ADDRESS OF 4K WORKAREA
         L     REG3,LEN4K              SIZE OF WORKAREA
         LR    REG4,REG2               ADDRESS OF 4K WORKAREA
         SR    REG5,REG5               CLEAR PADDING BYTE
         MVCL  REG2,REG4               INITIALIZE WORKAREA
*****************************************************************
*                                                               *
*        CHECK WHAT THE PARAMETER LIST PASSED US.               *
*                                                               *
*****************************************************************
         SPACE 1
         TM    M2SL_FLAGS,M2SL_MLWTO   MESSAGE IS MLWTO?
         BZ    SINGLINE                NO, SINGLE LINE...
         TM    M2SL_FLAGS,M2SL_MINORLINE  MINOR LINE?
         BZ    MAJRLINE                NO, MAJOR LINE...
*****************************************************************
*                                                               *
*        PROCESSING A MINOR WQE LINE.                           *
*                                                               *
*****************************************************************
MINRLINE EQU   *
         L     REG3,M2SL_WQE@          ESTABLISH ADDRESSABILITY
         USING WMJM,REG3               TO THE MAJOR WQE
         L     REG4,M2SL_WMNM@         ESTABLISH ADDRESSABILITY
         USING WMNM,REG4               TO THE MINOR WQE
*
*        DO STUFF NOW THAT WE KNOW THIS IS A MINOR WQE LINE.
*
         MVC   DYNAMTXT,WMNMTXT1       COPY MESSAGE TEXT
         DROP  REG3                    DROP ADDRESSABILITY
         DROP  REG4                    DROP ADDRESSABILITY
```

```
         TM    M2SL_FLAGS,M2SL_LASTLINE LAST LINE?
         BNZ   COMPLETE                 YES, MLWTO IS COMPLETE
         B     FINISHED                 PROCESSED MINOR, QUIT
*****************************************************************
*                                                               *
*        PROCESSING A MAJOR WQE LINE.                           *
*                                                               *
*****************************************************************
MAJRLINE EQU   *
         L     REG3,M2SL_WQE@           ESTABLISH ADDRESSABILITY
         USING WMJM,REG3                TO THE MAJOR LINE
*
*        DO STUFF NOW THAT WE KNOW THIS IS A MAJOR WQE LINE.
*
         MVC   DYNAMTXT,WMJMTXT         COPY MESSAGE TEXT
         DROP  REG3                     DROP ADDRESSABILITY
         TM    M2SL_FLAGS,M2SL_LASTLINE LAST LINE?
         BNZ   COMPLETE                 YES, MLWTO IS COMPLETE
         B     FINISHED                 PROCESSED MAJOR, QUIT
*****************************************************************
*                                                               *
*        PROCESSING A SINGLE WQE LINE.                          *
*                                                               *
*****************************************************************
SINGLINE EQU   *
         L     REG3,M2SL_WQE@           ESTABLISH ADDRESSABILITY
         USING WQE,REG3                 TO THE SINGLE WQE
*
*        DO STUFF NOW THAT WE KNOW THIS IS A SINGLE WQE LINE.
*
         MVC   DYNAMTXT,WQETXT          COPY MESSAGE TEXT
         DROP  REG3                     DROP ADDRESSABILITY
         B     COMPLETE                 MESSAGE IS COMPLETE
*****************************************************************
*                                                               *
*        WE HAVE PROCESSED A COMPLETE MESSAGE.                  *
*                                                               *
*****************************************************************
COMPLETE EQU   *
*
*        DO STUFF NOW THAT WE KNOW THE MESSAGE IS COMPLETE.
*
         B     FINISHED                 DONE PROCESSING, QUIT
*****************************************************************
*                                                               *
*        EXIT FROM THIS MODULE                                  *
*                                                               *
*****************************************************************
FINISHED EQU   *
         RETURN (14,12)                 RESTORE REGISTERS
         EJECT
*****************************************************************
*                                                               *
*        4K WORKAREA DEFINITIONS                                *
*        NOTE: THIS SAMPLE ONLY USES 128 BYTES OF THE WORKAREA *
*                                                               *
*****************************************************************
LEN4K    DC    F'4096'
DATAAREA DSECT
         DS    0H
DYNAMTXT DS    CL128                    MESSAGE TEXT AREA
DATAAVAL DS    CL3968                   AVAILABLE WORKAREA
         DS    0H
         ORG
DATAEND  EQU   *-DATAAREA
         SPACE 2
*****************************************************************
```

```
*                                                                     *
*        PARAMETER LIST MAPPING                                       *
*                                                                     *
***********************************************************************
         IHAWQE FORMAT=NEW             WTO QUEUE ELEMENT
         CNZMYM2S                      M2SL PARAMETER LIST
         EJECT
         END   M2SLEXIT
```

Chapter 4. CNZ_MSIEXIT — Master Scheduler Initialization Dynamic Exit

Topics for This Exit Appear as Follows:

Controlling the Exit Routine Through the Dynamic Exits Facilities

IBM has defined the CNZ_MSIEXIT exit to the dynamic exits facility. You can refer to the exit by the name CNZ_MSIEXIT. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

By default, the system disables the exit routine after one abend.

Exit Routine Environment

The exit routine receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31.
- With no locks or ENQs held.
- In the MASTER address space.

The exit receives control during Master Scheduler Initialization before the IEFSSNxx parmlib member is processed.

Exit Routine Processing

The exit routine is defined via the PROGxx parmlib member. Dynamic allocation is not available at this point in the IPL, so the exit routine must be defined in LPA, LNKLST or the NUCLEUS. The DSNAME parameter of the EXIT ADD statement in PROGxx should not be used for this exit routine. If it is used, the exit routine does not get control. For example, the PROGxx statement can be EXIT ADD EXITNAME(CNZ_MSIEXIT) MODNAME(MYMSIEXT).

Programming Considerations

You may code your CNZ_MSIEXIT to be reentrant or non-reentrant because the exit is invoked only once for the IPL.

Performance Considerations

While your CNZ_MSIEXIT has control, the IPL of the system does not continue. Therefore, keep the exit processing to a minimum.

Entry Specifications

No parameters will be passed by the exit point to the exit routine, while the 1-8 character parameter PARAM will be passed into the exit routine through Access Registers AR0 and AR1. The value of PARAM parameter is specified in the EXIT statement of the PROGxx parmlib member and in the SETPROG EXIT command.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0-12	Not applicable
13	Address of a 72 byte save area
14	Return address
15	Entry point address of the exit

Parameter Descriptions: Register 13 contains the address of a 72 byte save area on entry to the exit routine.

Access Registers AR0 and AR1 contain the data for the 1-8 character parameter PARAM, the value of which is specified in the EXIT statement of the PROGxx parmlib member and in the SETPROG EXIT command. AR0 contains the first 4 bytes, and AR1 contains the second 4 bytes. To get the full 8 bytes of PARAM data, the contents of AR0 and AR1 need to be concatenated.

Return Specifications

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-15	Restored to contents at entry

Chapter 5. CNZ_WTOMDBEXIT — WTO Message Data Block Exit

Topics for This Exit Appear as Follows:

Code a CNZ_WTOMDBEXIT exit routine when you want to view all messages being sent by WTO or WTOR. Information in the message is **readonly**; you can not modify the message contents.

CNZ_WTOMDBEXIT receives control from the system when a message is sent by a single-line WTO, a multi-line WTO, or a WTOR. Every single-line message that is sent by WTO or WTOR will be passed to the exit routines active at the exit point. Multi-line messages will be presented only when all lines have been completed. For example, a multi-line message with 1 major and 3 minor lines will result in the exit routine receiving control one time.

Controlling the Exit Routine Through the Dynamic Exits Facilities

IBM has defined the CNZ_WTOMDBEXIT exit to the dynamic exits facility. You can refer to the exit by the name CNZ_WTOMDBEXIT. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

1. The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error.
2. The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system disables the exit routine after one abend.

Exit Routine Environment

The exit receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31.
- With no locks or ENQs held.
- In the address space of the WTO or WTOR issuer, unless the message was a branch-entry message or was a multi-line message that timed out. In both of those cases, the exit receives control in the CONSOLE address space.

Exit Recovery: If the exit routine abnormally terminates, and the exit routine does not provide its own recovery, or the error percolates beyond the exit's recovery routine, a system recovery routine will get control.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

MVS invokes the CNZ_WTOMDBEXIT exit routine every time a WTO or WTOR single-line or completed multi-line message is sent. If any CNZ_WTOMDBEXIT routines are specified to the dynamic exits facility, a parameter list (mapped by macro CNZMYWMX) is passed that contains information about the message, such as:

- Whether the message is a single-line WTO
- Whether the message is a multi-line WTO
- Whether the message is a branch-entry WTO
- Whether the message is a WTOR
- First line of the message text, where the first 12 characters are typically the message identifier.
- Pointer to the message data block (MDB), mapped by macro IEAVM105

Also passed to CNZ_WTOMDBEXIT is the address of a 4K workarea for use by the exit routine.

Programming Considerations

1. All received messages are "completed" messages, in that control is received after the subsystem interface.
2. The exit routine will not affect the content of the message itself. The original message text and attributes (for example, routing and descriptor codes) remain intact. This exit routine cannot suppress a message.
3. Do not code an exit routine that receives control for a message that the exit issues; this causes an endless loop. The exit must be coded so that when it receives control for that message, it does not issue the message again.
4. If an exit routine needs to issue a message it should issue it as a branch entry message. Similarly, if an exit routine needs to DOM a message it should issue the DOM as branch entered.
5. Code the exit routine to be re-entrant.

Performance Considerations

WTO message processing can impact performance; therefore, consider the following recommendations so that system performance is not degraded:

1. Do not code an exit routine that contains an explicit or implicit WAIT or other processing of potentially long duration, for example, issuing requests for large amounts of dynamic storage, or issuing I/O requests. Exits that do this can adversely affect both application and overall system performance.
2. To reduce the need for storage requests, the system provides a 4K workarea that your exit routine can use for dynamic storage. The second word of the input parameter list points to the 4K workarea. The exit routine should clear what they put in the workarea before exiting, because all exit routines installed at the CNZ_WTOMDBEXIT exit point use the same workarea. Use this workarea for your program's working storage (for example, variables) and avoid doing a GETMAIN or STORAGE OBTAIN in the exit routine.
3. If recovery is necessary, set up an EUT functional recovery routine (FRR) instead of an ESTAE-type recovery routine for shortened path length of the exit routine. If it is not possible to use FRR recovery, then IEAARR should be used because it has the best set/delete performance for Estae-type recovery routines.

Entry Specifications

The system passes the address of the exit parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	**Contents**
0	Not applicable
1	Address of a pointer to the exit parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit

Parameter Descriptions: Register 1 points to the following:

Word 1
Address of the WMDX, mapped by macro CNZMYWMX

Word 2
Address of a 4K work area

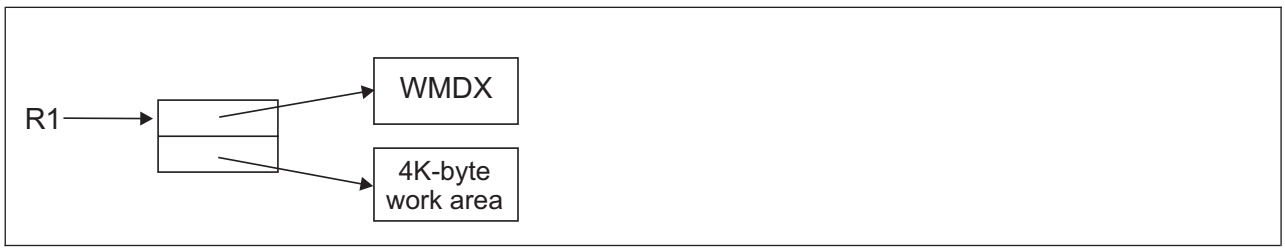

Figure 2. CNZ_WTOMDBEXIT input parameter structure

Return Specifications

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-15	Restored to contents at entry

Coded Example of the Exit Routine

The following is an example of an CNZ_WTOMDBEXIT exit:

```
             TITLE 'WMXEXIT - SAMPLE WTO MDB EXIT'
***START OF SPECIFICATIONS*************************************
*                                                             *
* MODULE NAME         =  WMXEXIT                              *
*                                                             *
* DESCRIPTIVE NAME    =  SAMPLE WTO MDB EXIT (FOR ACTIVATION  *
*                        ON CNZ_WTOMDBEXIT).                  *
*                                                             *
* FUNCTION            =  THIS EXIT DEMONSTRATES HOW TO        *
*                        USE THE DATA IN THE WMDX PARAMETER   *
*                        LIST TO IDENTIFY DIFFERENT TYPES OF  *
*                        MESSAGES.                            *
*                                                             *
*   OPERATION = THIS SAMPLE EXIT DETERMINES IF THE MESSAGE    *
*               DESCRIBED BY THE WMDX IS FOR A SPECIFIC       *
*               MESSAGE ID, A WTOR, SINGLE LINE MESSAGE, OR   *
*               MULTIPLE LINE MESSAGE AND PUTS TEXT IN THE    *
*               WMDX USER WORKAREA INDICATING WHAT IT FOUND.  *
*                                                             *
* NOTE1: THE WMDX PARAMETER LIST PROVIDES THE FIRST LINE OF   *
*       MESSAGE TEXT. IN ORDER TO OBTAIN SUBSEQUENT TEXT LINES*
*       OF A MULTI-LINE MESSAGE USE THE MDB POINTER (IN THE   *
*       WMDX) TO ACCESS THE MESSAGE DATA BLOCK (MDB), WHICH   *
*       CONTAINS DATA DESCRIBING THE MESSAGE, AND ALLOWS YOU  *
*       TO ACCESS THE SUBSEQUENT LINES. SEE "RECEIVING        *
*       MESSAGES AND COMMAND RESPONSES" IN "Z/OS MVS          *
*       ASSEMBLER SERVICES GUIDE", SA22-7608-08 FOR           *
*       INFORMATION ABOUT USING THE MDB. A GOOD EXAMPLE       *
*       PROGRAM USING THE MDB IS IEAEXMCS IN SAMPLIB.         *
*                                                             *
* ENTRY POINT         =  WMXEXIT                              *
*                                                             *
*   PURPOSE           =  DEMONSTRATE HOW TO USE THIS EXIT.    *
*                                                             *
*   LINKAGE = REGISTER 1 CONTAINS THE ADDRESS OF A PARAMETER  *
*             LIST, WHICH CONTAINS (1) THE ADDRESS THE WMDX   *
*             (MAPPED BY CNZMYWMX), AND (2) THE ADDRESS OF A  *
*             4K WORKAREA.                                    *
*                                                             *
* NOTE2: THERE IS ONLY ONE COPY OF THE WMDX AND WORKAREA      *
*       PER MESSAGE FOR ALL USERS OF CNZ_WTOMDBEXIT.  CARE    *
*       MUST BE TAKEN TO MAKE SURE THE WMDX DATA IS NOT       *
*       MODIFIED. THE WORKAREA IS NOT CLEARED BEFORE IT IS    *
*       PASSED TO SUBSEQUENT EXITS SO THE USER MUST CLEAR     *
*       IT IF IT CONTAINS SENSITIVE DATA.                     *
*                                                             *
*   INPUT DATA        =  REG1  ADDRESS OF THE PARAMETER LIST  *
*                        REG13 ADDRESS OF STANDARD SAVE AREA  *
*                        REG14 RETURN ADDRESS                 *
*                        REG15 ENTRY POINT ADDRESS            *
*                                                             *
*   REGISTERS SAVED   =  REG0 - REG15                         *
```

```
*                                                                     *
*   REGISTER USAGE    =  REG0  - USED FOR BASING                      *
*                        REG1  - PARAMETER REGISTER                   *
*                        REG2  - WORK REGISTER                        *
*                        REG3  - WORK REGISTER                        *
*                        REG4  - WORK REGISTER                        *
*                        REG5  - WORK REGISTER                        *
*                        REG6  - POINTER TO PARAMETER LIST            *
*                        REG7  - POINTER TO WMDX                      *
*                        REG8  - NOT USED                             *
*                        REG9  - NOT USED                             *
*                        REG10 - NOT USED                             *
*                        REG11 - POINTER TO 4K WORK AREA              *
*                        REG12 - MODULE BASE REGISTER                 *
*                        REG14 - RETURN POINT                         *
*                        REG15 - NOT USED                             *
*   REGISTERS RESTORED = REG0 - REG14                                 *
*                                                                     *
*   CONTROL BLOCKS    =                                               *
*     NAME     MAPPING MACRO   REASON USED           USAGE            *
*     ----     -------------   -----------           -------          *
*     WMDX     CNZMYWMX        EXIT PARAMETER LIST    R               *
*     WORKAREA NOT MAPPED      FOR USER               W               *
*                                                                     *
*   KEY = R-READ, W-WRITE, C-CREATE, D-DELETE                         *
*                                                                     *
*   TABLES            =  NONE                                         *
*                                                                     *
*   MACROS            =  NONE                                         *
*                                                                     *
*   MESSAGES          =  NONE                                         *
*                                                                     *
*   MODULE TYPE       =  CSECT                                        *
*                                                                     *
*   ATTRIBUTES      =  REENTRANT, REUSABLE, AMODE 31,                 *
*                        RMODE ANY                                    *
*                                                                     *
*01* DISCLAIMER =                                                    *
*                                                                     *
*  THIS SAMPLE PROGRAM IS PROVIDED FOR TUTORIAL PURPOSES ONLY.        *
*  A COMPLETE HANDLING OF ERROR CONDITIONS HAS NOT BEEN SHOWN         *
*  OR ATTEMPTED, AND THIS SOURCE HAS NOT BEEN SUBMITTED TO            *
*  FORMAL IBM TESTING. THIS SOURCE IS DISTRIBUTED ON AN               *
*  'AS IS' BASIS WITHOUT ANY WARRANTIES EITHER EXPRESSED OR           *
*  IMPLIED.                                                           *
*                                                                     *
***********************************************************************
         EJECT
WMXEXIT  CSECT
WMXEXIT  AMODE 31                     31-BIT ADDRESSING MODE
WMXEXIT  RMODE ANY                    31-BIT RESIDENCE
         SPACE 1
***********************************************************************
*                                                                     *
*         REGISTER ASSIGNMENTS                                        *
*                                                                     *
***********************************************************************
REG0     EQU   0                       REGISTER 0
REG1     EQU   1                       REGISTER 1
REG2     EQU   2                       REGISTER 2
REG3     EQU   3                       REGISTER 3
REG4     EQU   4                       REGISTER 4
REG5     EQU   5                       REGISTER 5
REG6     EQU   6                       REGISTER 6
WMDXPPTR EQU   6                       REGISTER 6 - PLIST
REG7     EQU   7                       REGISTER 7
WMDXPTR  EQU   7                       REGISTER 6 - WMDX ADDRESS
```

```
REG8     EQU   8                       REGISTER 8
REG9     EQU   9                       REGISTER 9
REG10    EQU   10                      REGISTER 10
REG11    EQU   11                      REGISTER 11
DATAREG  EQU   11                      REGISTER 11 - 4K WORKAREA ADDR
REG12    EQU   12                      REGISTER 12
BASEREG  EQU   12                      REGISTER 12 - MOD BASE
REG13    EQU   13                      REGISTER 13
REG14    EQU   14                      REGISTER 14
REG15    EQU   15                      REGISTER 15
*****************************************************************
*                                                               *
*        STANDARD ENTRY LINKAGE                                 *
*                                                               *
*****************************************************************
         SAVE  (14,12)                 SAVE REGISTERS
         BASR  BASEREG,REG0            ESTABLISH MODULE BASE
         USING *,BASEREG
         LR    WMDXPPTR,REG1           PARAMETER LIST
         SPACE 1
*****************************************************************
*                                                               *
*        ESTABLISH ADDRESABILITY                                *
*                                                               *
*****************************************************************
         L     WMDXPTR,0(,WMDXPPTR)    GET ADDRESSABILITY
         USING WMDX,WMDXPTR             TO WMDX
         L     DATAREG,4(,WMDXPPTR)    GET ADDRESSABILITY
         USING WORKAREA,DATAREG         TO 4K WORKAREA
         SPACE 1
*****************************************************************
*                                                               *
*        CHECK FOR A SPECIFIC MESSAGE. IN THIS EXAMPLE,         *
*        IEE114I (RESPONSE TO D A,L COMMAND) IS USED.           *
*                                                               *
*****************************************************************
         CLC   WMDX_MSGTEXT(8),=C'IEE114I ' IS THIS CORRECT MSG
         BNE   CHKWTOR                 NO, CONTINUE
*
*        DO STUFF NOW THAT WE KNOW THE MESSAGE IS CORRECT MSG
*
         MVC   WORKATXT(L'WORKATXT),=C'MESSAGE IS IEE114I.      '
         B     FINISHED                DONE
         SPACE 1
*****************************************************************
*                                                               *
*        CHECK FOR A WTOR                                       *
*                                                               *
*****************************************************************
CHKWTOR  EQU   *
         TM    WMDX_FLAGS,WMDX_WTOR    IS MESSAGE A WTOR
         BNO   CHKSLWTO                NO, CONTINUE
*
*        DO STUFF NOW THAT WE KNOW THE MESSAGE IS A WTOR
*
         MVC   WORKATXT(L'WORKATXT),=C'MESSAGE IS A WTOR.       '
         B     FINISHED                DONE
         SPACE 1
*****************************************************************
*                                                               *
*        CHECK FOR A SINGLE LINE MESSAGE                        *
*                                                               *
*****************************************************************
CHKSLWTO EQU   *
         TM    WMDX_FLAGS,WMDX_SLWTO   IS MESSAGE A SLWTO
         BNO   MULTLINE                NO, MSG IS A MULTILINE
*
```

```
*        DO STUFF NOW THAT WE KNOW THE MESSAGE IS A SINGLE LINE
*
         MVC   WORKATXT(L'WORKATXT),=C'MESSAGE IS A SINGLE LINE.'
         B     FINISHED                DONE
         SPACE 1
MULTLINE EQU   *
*****************************************************************
*                                                               *
*        PERFORM MULTIPLE LINE MESSAGE PROCESSING.              *
*        NOTE THAT WMDX_MLWTO COULD HAVE BEEN USED TO           *
*        DETERMINE IF MESSAGE IA A MULTI-LINE.                  *
*                                                               *
*****************************************************************
*
*        DO STUFF NOW THAT WE KNOW THE MESSAGE IS A MULTILINE
*
*        SEE NOTE ABOVE  ON HOW TO LOCATE THE REMAINING LINES IN
*        THE MDB.
*
         MVC   WORKATXT(L'WORKATXT),=C'MESSAGE IS A MULTI-LINE. '
         B     FINISHED                DONE
         SPACE 1
*****************************************************************
*                                                               *
*        EXIT FROM THIS MODULE                                  *
*                                                               *
*****************************************************************
FINISHED EQU   *
         RETURN (14,12)                RESTORE REGISTERS
         EJECT
*****************************************************************
*                                                               *
*        4K WORKAREA DEFINITIONS                                *
*        NOTE: THIS SAMPLE ONLY USES 25 BYTES OF THE WORKAREA   *
*                                                               *
*****************************************************************
WORKAREA DSECT
         DS    0H
WORKATXT DS    CL25                   MESSAGE TEXT AREA
WORKREST DS    CL4071                 AVAILABLE WORKAREA
         DS    0H
         ORG
WORKAEND EQU   *-WORKAREA
         EJECT
*****************************************************************
*                                                               *
*        PARAMETER LIST MAPPING                                 *
*                                                               *
*****************************************************************
         CNZMYWMX                     WMDX PARAMETER LIST
         SPACE 2
         END   WMXEXIT
```

Chapter 6. CSVLLIX1 — LLA Module Fetch Exit

Topics for This Exit Appear as Follows:

Library lookaside (LLA) improves the performance of fetching modules from both LNKLST and non-LNKLST data sets, and is a control point in managing updates to these data sets on DASD. For each module that is fetched, LLA dynamically accumulates statistics such as the fetch rate and the fetch durations. Using these and other statistics, LLA periodically triggers LLA module staging analysis to evaluate the cost of fetching each module. Based on projected savings, LLA module staging analysis places copies of the most frequently used modules into a virtual lookaside facility (VLF) data space. LLA can then fetch these modules from virtual storage without I/O and with a reduced number of processor instructions.

LLA module fetch keeps track of the number of modules fetched from the LLA libraries. When a default threshold of 2000 module fetches from a library is reached, or after the initial 10 fetches of a single module from DASD, LLA module fetch triggers the LLA module staging analysis function.

Each time LLA fetches a module from an LLA library, it logs statistics and then calls the installation exit CSVLLIX1. LLA fetch passes to CSVLLIX1 the address of a parameter list containing fetch statistics, the address of a user work area, and a copy of the module's BLDL format PDS directory entry.

You can use CSVLLIX1 to:

- Monitor and collect fetch statistics.
- Control the 2000 fetch default limit.
- Cause staging analysis to begin regardless of the statistics for one module or all modules.
- Modify statistics to influence the triggering of staging analysis.

Controlling The Exit Routine Through the Dynamic Exits Facility

IBM has defined the CSVLLIX1 installation exit to the dynamic exits facility. You can refer to the exit by the name CSVLLIX1. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and the exit routines of the exit.

When you start LLA, if the EXIT1(OFF) statement is not present in the CSVLLAxx parmlib member being used by LLA, the system attempts to add exit routine CSVLLIX1 in the following situations:

- no exit routines have been associated with CSVLLIX1 by PROGxx
- no exits have been associated using SETPROG
- there are no exit routines from a previous LLA start

If you have associated exit routines with CSVLLIX1 the system does not use the default exit routine. In this case, if you require an exit routine of the default name, you must explicitly add the default exit routine to PROGxx.

A change from EXIT1(ON) to EXIT1(OFF) has no effect. If you want to deactivate the CSVLLIX1 exit routine (or any other exit routine for the CSVLLIX1 exit), you can use the PROGxx EXIT statement with SET PROG=xx or you can use SETPROG EXIT.

To limit the number of times the exit routine ends with an abend, before the exit routine becomes inactive, you can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro, or the ABENDNUM parameter of the SETPROG EXIT operator command. The system disables the exit routine if the exit routine ends with an abend on two successive calls. An abend is encountered when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error.
- The system allows a retry (the recovery routine is entered with bit SDWACLUP off).

The system disables the exit routine if the exit routine ends with an abend, as defined above, on two successive calls.

Exit Routine Environment

CSVLLIX1 receives control in the following environment:

- Enabled for interrupts.
- In supervisor state and in primary ASC mode with the primary ASID equal to the home ASID.
- In AMODE 31 and RMODE ANY.
- Under a content supervisor's SVRB within the user's address space.
- With no locks or ENQs held.
- Under any task that might issue a LINK, LOAD, XCTL, or ATTACH macro.

Exit Recovery: The LLA recovery routine protects the exit routines of CSVLLIX1. The recovery routine records diagnostic information in the system diagnostic work area (SDWA), requests recording of the error in the logrec data set, and takes an SVC dump. LLA then retries to a point within LLA module fetch to call the next exit routine.

Exit Routine Processing

When a caller issues a LINK, LOAD, XCTL, or ATTACH macro, the system must fetch the module if it doesn't already reside in virtual storage. If the module is in a data set that LLA is managing, then an LLA fetch occurs. If the module has already been staged, then LLA fetch obtains the module from a VLF data space to avoid program fetch I/O. If the module has not been staged, LLA fetch obtains the module from DASD. LLA first logs statistics about the fetch and then passes control to CSVLLIX1. The exit routine examines the parameters and returns a code indicating whether LLA should trigger module staging analysis.

The LLP1USER field in LLP1, the exit parameter list, contains a 31-bit address that points to a 4-byte user data area in the CSA. The user data area is aligned on a fullword boundary and is initialized to 0. The user data area is also pointed to by the LLP2X1US field in the CSVLLIX2 (LLA module staging exit) parameter list. An installation can use the user data area as a means of allowing CSVLLIX1 and CSVLLIX2 to share information. For example, CSVLLIX1 or CSVLLIX2 can acquire storage in the CSA (by issuing a GETMAIN or STORAGE macro) and place the address of this storage in the 4-byte user data area. CSVLLIX1 could write records to the common storage area during fetch time and CSVLLIX2, when it gets control, could then write the records to DASD. This action could reduce the time CSVLLIX1 needs for processing and thus shorten LLA's path length for fetching a module.

CSVLLIX1 can also use the LLP1USER field to pass information (such as a parameter or the address of a parameter) to itself during subsequent invocations of the exit routine.

If your installation plans to have both CSVLLIX1 and CSVLLIX2 access the 4-byte user data area (pointed to by LLP1USER and LLP2X1US), both exits must manage the serialization of the area. If only CSVLLIX1 is going to use the area then CSVLLIX1 must manage the serialization of the user data area. Compare-and-swap (CS) is a potential serialization method. The 4-byte user data area is initialized to 0. If LLA is restarted, the user data area is not reset to 0. It will contain the last value stored in it by either CSVLLIX1 or CSVLLIX2. When using this field as a counter, you must reset it when the condition you are testing for is met.

Programming Considerations

Code CSVLLIX1 to be reentrant.

CSVLLIX1 is called each time a program fetch occurs for LLA managed members. When coding a modification to the exit, you should be aware that an increased path length will increase processor utilization. If your installation wants to limit its use of CSVLLIX1, perhaps to improve performance by shortening path length, you can deactivate the exit.

Note: Be aware while coding the exit routine, that because the exit is called for every module fetch request, an increased path length increases processor utilization which can degrade system performance.

Macro Instructions and Restrictions: While CSVLLIX1 has control, it cannot issue (or cause another program to issue) a LOAD, LINK, XCTL, or ATTACH macro that might require LLA to fetch a module from a data set that LLA is managing. This would result in recursive calls to the exit routine. See *z/OS MVS Initialization and Tuning Guide* for information on managing a data set.

Note: Do not code the exit routine to issue the WAIT macro or call a service, such as WTOR, that issues a WAIT. Do not code the exit routine to call a service that suspends processing for a period of time, like STIMER(M), which will also have a severely negative impact upon system performance.

I/O Restrictions: Some functions that are needed to perform I/O, such as dynamic allocation, must be initialized.

Entry Specifications

LLA passes to CSVLLIX1 the address of the LLP1 parameter list.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Zero
1	Address of the LLA module fetch exit parameter list (LLP1)
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of CSVLLIX1

Parameter List Contents: Register 1 contains the address of LLP1, the LLA fetch exit parameter list. The macro IHALLP1 maps LLP1. The LLP1 mapping is described in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

CSVLLIX1 passes back to LLA both a return code and a reason code to indicate whether LLA should trigger module staging analysis.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0	One of the following reason codes:

Reason Code	Explanation
0	Use the default threshold to trigger LLA module staging analysis.
1	Indicates that LLA module staging analysis should begin.
2	Indicates that LLA module staging analysis should not begin.

1-14	Restored to contents at entry
15	One of the following return codes:

Return Code	Explanation

0 Indicates that the default threshold should be used to trigger LLA module staging analysis. The reason code in register 0 should be zero.

4 Indicates that the exit has determined whether to trigger LLA module staging analysis. The reason code in register 0 provides additional information.

Coded Example of the Exit Routine

A sample CSVLLIX1 exit routine is provided in SYS1.SAMPLIB for your reference. This routine contains the code for entry to the exit routine (a standard BR14 branch).

Chapter 7. CSVLLIX2 — LLA Module Staging Exit

Topics for This Exit Appear as Follows:

Library lookaside (LLA) improves the performance of fetching modules from both LNKLST and non-LNKLST data sets, and is a control point in managing updates to these modules on DASD. LLA collects statistics such as fetch rates and fetch durations to allow LLA module staging to determine the value of staging each module. LLA module staging runs periodically to perform this evaluation and stages, or places, copies of the most frequently used modules into a virtual lookaside facility (VLF) data space. LLA can then fetch these selected modules from virtual storage without I/O and with a reduced number of processor instructions.

LLA combines four independent components of staging value to determine the net value of staging a module into processor storage (see "Exit Routine Processing" on page 41). LLA multiplies each component of staging value by its weighting factor, adds the products, and determines the net LLA staging value for the module. LLA stages modules that have the highest LLA staging value. Modules with a low LLA value are provided to users through program fetch from DASD.

Before applying the weighting factors to determine whether a module should be staged, LLA calls CSVLLIX2 and passes it the address of a parameter list that contains the weighting factors. CSVLLIX2 can modify the weighting factor values or specify an installation-defined value and thus determine which modules are staged.

You can use CSVLLIX2 to:

- Analyze fetch statistics provided in the LLA module staging parameter list (LLP2).
- Influence the calculation of the LLA value (which determines if a module should be staged) by altering the weighting factors in the LLP2 parameter list.
- Force LLA to stage the module by setting the appropriate return and reason codes.

Controlling The Exit Routine Through the Dynamic Exits Facility

IBM has defined the CSVLLIX2 installation exit to the dynamic exits facility. You can refer to the exit by the name CSVLLIX2. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and the exit routines of the exit.

When you start LLA, if the EXIT2(OFF) statement is not present in the CSVLLAxx parmlib member being used by LLA, the system attempts to add exit routine CSVLLIX2 in the following situations:

- no exit routines have been associated with CSVLLIX2 by PROGxx
- no exits have been associated using SETPROG
- there are no exit routines from a previous LLA start

If you have associated exit routines with CSVLLIX2 the system does not use the default exit routine. In this case, if you require an exit routine of the default name, you must explicitly add the default exit routine to PROGxx.

A change from EXIT2(ON) to EXIT2(OFF) has no effect. If you want to deactivate the CSVLLIX2 exit routine (or any other exit routine for the CSVLLIX2 exit), you can use the PROGxx EXIT statement with SET PROG=xx or you can use SETPROG EXIT.

To limit the number of times the exit routine ends with an abend, before the exit routine becomes inactive, you can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro, or the ABENDNUM parameter of the SETPROG EXIT operator command. The system disables the exit routine if the exit routine ends with an abend on two successive calls. An abend is encountered when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error.
- The system allows a retry (the recovery routine is entered with bit SDWACLUP off).

The system disables the exit routine if the exit routine ends with an abend, as defined above, on two successive calls.

Exit Routine Environment

CSVLLIX2 receives control in the following environment:

- Enabled for interrupts.
- In supervisor state and in primary ASC mode with the primary ASID equal to the home ASID.
- In AMODE 31 and RMODE ANY.
- With no locks or ENQs held.
- In task mode under a non-jobstep TCB attached by LLA's jobstep program in LLA's address space.
- With TCB and PSW keys of zero.

Exit Recovery: The LLA recovery routine protects the exit routines of CSVLLIX2. The recovery routine records diagnostic information in the system diagnostic work area (SDWA), requests recording of the error in the logrec data set, and takes an

SVC dump. LLA then retries to a point within LLA staging to call the next exit routine.

Exit Routine Processing

CSVLLIX2 receives control from the LLA staging function before it applies the weighting factors to determine a module's total staged value. LLA staging combines 4 independent staging values to determine the net value of staging a module into processor storage. The 4 values, which are contained in field LLP2VALU of the parameter list, relate to response time, processor storage, contention, and an optional, installation-defined cost. Each staging value is in the range -10,000 to +10,000 and indicates the relative value to the system of LLA staging the module. Each value has a corresponding weighting factor (LLP2WGTS) in the range 0 to 100, used to indicate the relative importance of the value. LLA staging multiplies each factor of staging value by its weighting factor, adds the products, and determines the net LLA staging value for the module.

CSVLLIX2 receives control from LLA staging before it applies the weighting factors to determine whether a module should be staged. CSVLLIX2 can influence the calculation of the module's LLA staging value by altering the appropriate weighting factors. For example, if response time is more important to your installation than processor storage use, you can use CSVLLIX2 to set the response time weighting factor to a high value (70-100) and set the processor storage weighting factor to a low value (0-30).

CSVLLIX2 can introduce its own installation-defined value (LLP2VUSR). The value must be in the specific range (-10,000 to +10,000), and the corresponding weighting factor must be changed from its initial value of zero.

CSVLLIX2 can also force a module to be staged by setting the appropriate return and reason codes. This is a less desirable solution because LLA can then change the staging only at the direction of CSVLLIX2; for example, it could never deactivate the staged module even if it was so infrequently used that it was stolen to auxiliary storage.

The LLP2X1US field in LLP2, the exit parameter list, contains a 31-bit address that points to a 4-byte user data area in the CSA. The user data area is aligned on a fullword boundary and is initialized to 0. The user data area is also pointed to by the LLP1USER field in the CSVLLIX1 (LLA module fetch exit) parameter list. An installation can use the user data area as a means of allowing CSVLLIX1 and CSVLLIX2 to share information. For example, CSVLLIX1 or CSVLLIX2 can acquire storage in the CSA (by issuing a GETMAIN or STORAGE macro) and place the address of this storage in the 4-byte user data area. CSVLLIX1 could then write records to this common storage area during fetch time and CSVLLIX2, when it gets control, could then write the records to DASD. This action could reduce the time CSVLLIX1 needs for processing and thus shorten LLA's path length for fetching a module.

If your installation plans to have both CSVLLIX1 and CSVLLIX2 access the 4-byte user data area (pointed to by LLP1USER and LLP2X1US), both exits must manage the serialization of the area. If only CSVLLIX1 is going to use the area, then CSVLLIX1 must manage the serialization of the user data area. Compare-and-swap (CS) is a potential serialization method. The 4-byte user data area is initialized to 0. If LLA is restarted, the user data area is not reset to 0. It will contain the last value stored in it by either CSVLLIX1 or CSVLLIX2. When using this field as a counter, you must reset it when the condition you are testing for is met.

The LLP2USER field in LLP2, the exit parameter list, points to a 4-byte user data area that is aligned on a fullword boundary and is reserved for CSVLLIX2 to use. CSVLLIX2 can use the 4-byte user data area as a work area to pass information to itself during subsequent invocations. The 4-byte user data area is initialized to zero and subsequently contains any value stored in it by CSVLLIX2. When using this field as a counter, you must reset it when the condition you are testing for is met.

In most cases LLA does apply the same algorithm to program objects, however in a few situations LLA finds staging in VLF unacceptable due to certain attributes of program objects. For example, if LLA discovers that the number of deferred classes in the header of the program object is not 0 or the program object is a split RMODE module, then LLA cancels the staging algorithm, making a particular program object ineligible for staging.

Programming Considerations

Code the exit routine to be reentrant. There are no restrictions on external routines that CSVLLIX2 can invoke while it has control.

If your installation wants to limit its use of CSVLLIX2, perhaps to improve performance by shortening path length, you can deactivate the exit. CSVLLIX2 is ON by default.

Changing Weighting Factors: The default weighting factors that CSVLLIX2 can change are set to response time=75, contention=50, storage=25, and installation=0. You should not need to change these settings. However, if an adjustment is necessary, consider the following:

- Response time (LLP2WRSP)

  Keep this weighting factor higher than the others if you have sufficient central storage to hold all the staged modules. Decrease this slightly relative to the storage weighting factor if LLA quickly fills the VLF data space and the data space cannot be enlarged. (To enlarge the data space, increase the value specified by the MAXVIRT keyword for class CSVLLA in the COFVLFxx parmlib member.) Processor time might increase if many staged modules age out to expanded storage because of central storage contention. In general, for the modules that remain backed by central storage, LLA uses slightly less processor time than program fetch for all module sizes.
- Contention (LLP2WCTN)

  Increase this factor relative to response time if data sets that LLA is managing incur I/O activity that causes disruptive contention. Such data sets are likely to be on volumes shared between systems or on volumes that contain other performance-sensitive data sets. You can use storage isolation to control contention for LLA's VLF data space.
- Storage (LLP2WSTO)

  Increase this factor if you do not have enough processor storage to hold all the modules. The storage value should always be weighted lower than the response time value because the storage value usually is negative.

The values that LLA calculates using these weighting factors are derived as follows:

- Response time (LLP2VRSP)

  This value is defined as the time that would be saved per sample if the module were to be fetched from VLF divided by the total time that would have been saved during the previous sample if all modules had been fetched from VLF.

- Contention time (LLP2VCTN)

  This value is defined as the average difference between the minimum fetch for the module from LLA or DASD, and the duration of the fetch. The average LLA difference is then subtracted from the average DASD difference. The result is multiplied by the number of fetches, and is normalized by dividing by the sum of all the contention deltas from the previous sample.

  Contention value helps LLA to compensate dynamically for unbalanced storage and I/O resources in a system.
- Storage (LLP2VSTO)

  This value is defined as the number of processor storage bytes that would be saved minus the number of bytes that would be spent if the module were to be kept in VLF, normalized by dividing the sum of all the storage deltas accumulated in the previous sample. Storage is saved because users occupy storage for less time while they wait for fetches. Storage is spent because the staged module requires processor storage to back it in the VLF data space. With the exception of some very small and highly used modules, storage value is a negative number.

Changing the Staging Threshold: If your installation has a steady state workload, you should notice a consistency in the modules being staged and in the ranking of the modules. If the same modules are not being staged, or if the ranking of the modules staged fluctuates often, the reason could be that your installation has a large number of modules in its working set and therefore, LLA needs to have a larger sampling of modules on which to base its staging decisions.

Increase the staging threshold (the default is 2000) to provide LLA with a more meaningful sampling on which to base its decisions, so that LLA will consistently stage the most often used modules. It is suggested that the staging threshold be at least 3 times the total number of modules in your installation's working set.

Using the Exit to Influence Module Staging: If you use CSVLLIX2 to force LLA to stage a particular module, the change applies only to the current staging analysis of the module. That is, LLA does not maintain the changed staging status of the module across subsequent exit invocations. In addition, if the exit is deactivated or the code of the exit routine changed, the outcome of staging analysis might be affected.

Depending on your installation's workload, the staging analysis for a given module might be different each time CSVLLIX2 is invoked. If you use CSVLLIX2 to determine whether a particular module should be staged, the exit routine should continue to check the module's staging statistics during subsequent invocations.

If you plan to use CSVLLIX2 to force LLA to stage a particular module regardless of the module's staging statistics, the exit routine should maintain this status each time the exit is invoked. Do this by setting the appropriate return and reason codes. Otherwise, excessive LLA overhead can occur.

For example, avoid using program logic that indicates that a module, if not staged, should be staged, but fails to indicate an action if the module is already staged. In this case, when the exit fails to indicate an action, LLA staging analysis will make its own determination and might deactivate staging for the module. The next invocation of the exit for this module would cause LLA to restage the module, and so on. Excessive LLA overhead results as LLA alternately stages and deactivates the same module repeatedly.

Entry Specifications

LLA passes to CSVLLIX2 the address of the LLP2 parameter list.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Zero
1	Address of the LLA staging installation exit parameter list (LLP2)
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of CSVLLIX2

Parameter List Contents: Register 1 contains the address of LLP2, the LLA staging exit parameter list. The macro IHALLP2 maps data area LLP2. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for a mapping of the LLP2 data area.

Return Specifications

CSVLLIX2 passes back to LLA both a return code and a reason code to determine the need for LLA module staging.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0	One of the following reason codes (the reason code in register 0 is dependent upon the return code in register 15):

Reason Code	Explanation
0	Indicates that LLA should use the weighting factors to determine whether the module should be staged. (Set register 15 to 0.)
1	Indicates that LLA is to stage this module. If the module is already staged, it remains staged. (Set register 15 to 4.)
2	Indicates that LLA is not to stage this module. If the module is already staged, LLA stops using the staged copy of the module. (Set register 15 to 4.)

1-14	Restored to contents at entry
15	One of the following return codes:

Return Code	Explanation
0	Indicates that LLA module staging should use the weighting factors to determine whether the module should be staged.

4 Indicates that LLA module staging should not use the weighting factors. The decision to stage or not stage the module is specified by the reason code in register 0.

Coded Example of the Exit Routine

IBM provides a sample CSVLLIX2 exit routine in SAMPLIB for your reference. This routine contains the code for entry to the exit routine (a standard BR14 branch).

Chapter 8. DLF Connect / Disconnect Exit

Topics for This Exit Appear as Follows:

This exit provides the control information that hiperbatch and the data lookaside facility (DLF) need.

Hiperbatch is a performance enhancement that can significantly reduce the execution time of certain batch job streams or multi-step batch jobs that access the same VSAM or QSAM data sets. Hiperbatch works with DLF to allow batch jobs to share access to a DLF object. A DLF object is a set of hiperspaces created by DLF that contains QSAM or VSAM data managed by hiperbatch. Storing data into, and retrieving data from, DLF objects is done transparently by the access method; an installation does not need to rewrite its application programs or the JCL needed to run them. The installation does, however, need to provide control information to DLF.

If your system includes RACF 1.9 or higher, you can define RACF profiles in the DLFCLASS general resource class instead of using this exit routine. However, you must still code the exit; its decisions can override the information in the RACF profiles.

If your system *does not* include RACF 1.9 or higher, you must code a DLF Connect / Disconnect installation exit routine to provide the needed control information.

You can use the DLF Connect / Disconnect exit to:

- Decide whether a job is eligible to connect to a DLF object. (The object will be created if this is the first connection.)

- Specify whether or not to retain a DLF object even when no jobs are connected to it. The default is to delete the DLF object when there are no longer any connections to it.
- Determine whether a particular data set is eligible for DLF processing, so that if the data set is updated while DLF is not running, DLF will process the updated form of the data set. (See "Query" on page 50.)

To make these decisions, the exit must have access to the DLF control information:

- The names of all data sets that you want the system to process as DLF objects
- The names of all DLF objects that are to be retained
- The users and/or job names that are allowed to access each DLF object

You can store this control information in a data set or code the information in the exit routine. If you have a great deal of control information, or if you plan to update it often, you should consider storing the information in a data set.

Installing the Exit Routine

Before starting the Data Lookaside Facility:

1. You must include the exit in an authorized library in the LNKLST concatenation.
2. You must specify the exit name on the CONEXIT keyword in the COFDLFxx parmlib member.

If your installation runs jobs that perform random updates of VSAM or QSAM data sets while other jobs read the data sets, you must name your exit COFXDLF1. Hiperbatch calls this name when DLF is not running to request the exit to query VSAM or QSAM data sets. If your installation does not access VSAM or QSAM data sets when DLF is not running, you may choose any name for the exit. (See "Query" on page 50.)

If you name the exit routine COFXDLF1—specify CONEXIT(COFXDLF1) in the COFDLFxx member in the parmlib—the DLF initialization code will look for that routine in the LPA. If it is not there, it will load the exit routine into the CSA from the LNKLST. Then, as long as the DLF is active, the system will call the exit via a branch to the LPA or CSA address, avoiding a LINK or LOAD.

Note: If you haven't named your exit routine on the CONEXIT keyword in the COFDLFxx parmlib member, the system does a BLDL to the LNKLST for the exit; when BLDL finds the routine, the system issues a LINK with DE= without first searching the LPA. To avoid this overhead where your installation makes infrequent accesses to the same VSAM or QSAM data sets and thus does not need to use hiperbatch, simply don't start the DLF.

You cannot replace the exit while the DLF is active. The exit that is requested when the DLF is started remains in effect for the duration of the DLF address space. To replace the exit, you must stop the DLF, replace the exit (or change the parmlib CONEXIT parameter to point to a different exit), and then start the DLF again.

For general instructions on installing an exit routine, see "Link editing an Installation Exit Routine into a Library" on page 3. For information on how to start the DLF, see *z/OS MVS System Commands*.

Exit Routine Environment

The DLF exit receives control in the following environment:

- Resides in an APF-authorized library in the LNKLST concatenation
- Enabled for interrupts
- In primary ASC mode
- In AMODE 31 and RMODE ANY
- In supervisor state with PSW key 0
- When called to perform the initialization function, the exit is invoked in the DLF address space. On subsequent calls, in which the routine performs the query, connect, and disconnect functions, the exit routine runs in the address space of the task that issues the request

Exit Recovery

DLF recovery protects the exit.

If the exit abnormally terminates during DLF initialization, the DLF initialization is terminated.

If the exit abnormally terminates during a query call, DLF determines that the data set is not eligible for DLF processing. OPEN processing continues and the job reads the data set from DASD, instead of from a DLF object, and updates the data set on DASD.

If the exit abnormally terminates during a user's attempt to connect to a DLF object, the request is terminated and the user is not connected to the DLF object. OPEN processing continues and the job reads the data set from DASD, instead of from the DLF object.

If the exit abnormally terminates during a user's attempt to disconnect from the DLF object, the user is disconnected from the DLF object.

Exit Routine Functions

The exit is called to perform these functions: initialization, query, connect, and disconnect.

Initialization

When the DLF is started, the exit is invoked with a request from the DLF to perform initialization: the CXTFUN field in the CXT (the exit routine parameter list) is set to 0. If your installation places control information in a data set, use the initialization call to access the data set. On subsequent calls, the exit routine uses the control information in the data set to perform the query, connect, and disconnect functions.

In the CXT, the CXTUDAB field points to a 16-byte area in the CSA. The exit routine can acquire storage in the CSA (for example, by issuing a GETMAIN or STORAGE macro) and place the address of this storage in the first word of a 16-byte area pointed to by CXTUDAB. The exit can then open the data set that contains the DLF control information and read it into the CSA area. DLF passes the

pointer to the address of the control information to the exit (in CXTUDAB) every time it is called. The exit now has access to the control information it will need on subsequent calls.

The exit must manage serialization of the 16-byte area in the CSA to prevent the address of the control information from being reset by the exit on a DLF restart while the area is being accessed for a query. An installation can serialize the 16-byte area by, for example, using the third word of the area as a serialization field and using compare-and-swap (CS) as a serialization method. If DLF is restarted, the 16-byte area is not reset to zero. It will contain the last values stored by the exit.

You do not need to code the exit to perform initialization if you code the control information within the exit routine itself. If you code the control information within the exit routine, your exit needs only to return control to DLF on an initialization call. The exit return code is ignored on an initialization call.

Query

If your installation runs jobs that perform random updates of VSAM or QSAM data sets while other jobs read the data sets, you must:

- Code your exit routine to perform the query function.
- Name your exit COFXDLF1.

If you do not code the exit routine to perform the query function, you can choose any name for the exit.

Hiperbatch calls the exit with a query request (the CXTFUN field in the CXT is set to 1) to determine whether or not a particular VSAM or QSAM data set that is being opened for update or output could be processed as a DLF object. The exit routine searches the installation's control information (its address is pointed to by CXTUDAB) for the name of the data set (in CXTDSN) and, optionally, its volume serial (in CXTVOL). If the data set is listed in the control information, the exit routine sets return code 0 to indicate that the data set is eligible for DLF processing. If the data set is not listed in the control information, the exit routine sets return code 8 to indicate that the data set is not eligible for DLF processing.

If the data set is listed in the control information, hiperbatch records the name of the data set to ensure that if a data set is opened while DLF is down, and DLF is started before the data set has been closed, subsequent jobs that open the data set will connect to its updated form.

It is strongly recommended that you start DLF before starting JES. This action is necessary to ensure that readers of VSAM and QSAM data sets read the latest level of the data sets.

Connect

DLF calls the exit when a job opens a VSAM or QSAM data set (the CXTFUN field in the CXT is set to 2). At this time, the exit routine must determine whether the user or job requesting the connection is eligible to connect to the DLF object. The exit also must specify whether the DLF object is to be retained. (If RACF profiles exist for the DLF objects, the exit routine might not need to make these decisions, but it can override the RACF information.) The exit sets a return code that indicates whether the connection is to be permitted, or whether RACF information to is to be used to make the decision.

If DLF is not active, connect processing always invokes exit COFXDLF1.

Disconnect

DLF calls the exit when it disconnects a user from a DLF object (the CXTFUN field in the CXT is set to 3). The exit return code is ignored on a disconnect call.

Exit Routine Processing

When a user attempts to open a VSAM or QSAM data set, and DLF is active, the system tries to connect the user to the DLF object corresponding to the data set.

The exit is only called after RACF or another security product has already granted the user access to the VSAM or QSAM data set. Then, on a system with RACF 1.9 or higher, DLF will check the job or user's eligibility to connect to the DLF object. If RACF approves the request, DLF updates CXTSFLGS in the CXT, the exit parameter list, with the connect information. Then DLF passes the CXT to the exit routine, which can override the RACF decision to connect to a DLF object.

On a system without RACF 1.9, the CXTSFLGS bits are set to zero, and the exit routine must determine whether the application is eligible to connect to a particular DLF object.

After exit routine processing, the exit returns a code specifying whether or not the user is allowed to connect to the DLF data object. The exit routine can override the eligibility of the DLF data object and inform DLF not to connect the DLF object to the user. The exit also sets bit CXTRTAIN in CXTUDATA to '1'B if the DLF object is to be retained when no jobs are connected to it. CXTRTAIN will be honored only if the data set is being opened for update (QSAM) or load (VSAM) mode.

For more information on the return codes the exit routine can set, see “Registers at Exit” on page 53.

Programming Considerations

If your installation does not include RACF 1.9 or higher (or if you are using the exit to override RACF decisions), the exit must determine which connections will be permitted. The exit must have access to the following:

- The names of all data sets that you want the system to process as DLF objects
- The names of all DLF objects that are to be retained
- The users and/or job names that are allowed to connect to each DLF object.

The exit routine must be reentrant and must reside in an APF-authorized library in the LNKLST concatenation.

If your installation runs jobs that perform random updates of VSAM or QSAM data sets while other jobs read the data sets, you must name your exit COFXDLF1. Hiperbatch calls this name when DLF is not running to request the exit to query VSAM or QSAM data sets. If your installation does not access VSAM or QSAM data sets when DLF is not running, you may choose any name for the exit (see “Query” on page 50). However, you must specify the name in the COFDLFxx parmlib member with the keyword CONEXIT, so that it is known to DLF.

The installation determines the maximum number of VSAM/QSAM data sets for which DLF objects may exist, based on available extended storage, and places the

value in the CXTDSMAX field. (The default is 50.) This value is read only on the first query or connect call to the exit. It is recommended that you code the exit to store the proper value in CXTDSMAX on all query and connect calls to the exit.

When the DLF is active, the exit is called for every OPEN (connect) and every CLOSE (disconnect) issued by a VSAM or QSAM application. If you plan to have the exit routine search very long lists of names, you should consider using a more efficient search technique than sequential (for example, a binary search).

Macro Instructions and Restrictions

With one exception, the exit routine cannot issue (or cause another program to issue) the OPEN macro, because the exit is already running under OPEN processing. The exception is the initialization function; during this call, the exit routine can issue the OPEN macro.

Entry Specifications

DLF passes to the exit the address of a fullword that points to the CXT, the exit parameter list.

Registers at Entry

The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of a fullword that points to the CXT, which is mapped by COFZCXIT
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit routine

Parameter List Contents

Register 1 contains the pointer to the fullword that contains the address of the CXT, the exit parameter list. The CXT is described in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

There is no field in the CXT that indicates whether or not a particular connection is allowed. The exit must set a return code on the connect exit call to communicate this information to DLF.

Note:

1. The exit routine indicates whether or not an object is to be retained by setting the CXTRTAIN bit on the connect exit call.
2. To include the CXT mapping in your exit routine, you must code 'COPY COFZCXIT' in the routine.

Return Specifications

A return code from the exit indicates whether the DLF object is eligible to be connected to the user (query), or whether the DLF object should be connected to the user (connect).

The exit routine indicates whether or not an object is to be retained by setting the CXTRTAIN bit on the connect exit call.

Registers at Exit

Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-1 Irrelevant

2-14 Same as at entry

15 One of the following return codes:

Query Return Code
Explanation

0 The data set is eligible for DLF processing.

4 DLF is to use information from the security product (e.g., RACF) to determine whether the data set is eligible for DLF processing.

8 The data set is not eligible for DLF processing.

Connect Return Code
Explanation

0 DLF is to permit the user to connect to the DLF object.

4 DLF is to use information from the security product (e.g., RACF) to determine whether the user is authorized to connect to the DLF object. If RACF 1.9 or higher is not included in the system, DLF is not to permit the user to connect to the DLF object.

8 DLF is not to permit the user to connect to the object.

Coded Example of the Exit Routine

IBM provides two coded examples of the DLF Connect / Disconnect installation exit routine in SYS1.SAMPLIB. These are COFXDLF1 and COFXDLF2.

COFXDLF2 requires the use of two routines, COFXUPDT and COFXMACS, both of which are also provided in SYS1.SAMPLIB.

Chapter 9. HIS.SERVSTAT— HISSERV Service Exit

Topics for This Exit Appear as Follows:

The HIS.SERVSTAT exit will be defined by IBM. Any authorized program that wants to know the state of the HISSERV service can register with the HIS.SERVSTAT dynamic exit. For example, an exit can provide a bootstrap process for potential exploiters of the HISSERV service, as a mechanism to know when a program is able to begin exploiting the service. Exit routines are called for the following reasons:

- The HISSERV service has been enabled.
- The HISSERV service has started instrumentation data collection.
- The HISSERV service has stopped instrumentation data collection.
- The HISSERV service had been disabled.

The HISYEXIT macro maps the storage passed to a HIS exit routine that is monitoring the service.

Controlling the Exit Routine Through the Dynamic Exits Facility

The exit routine is defined by registering an exit routine with the HIS.SERVSTAT dynamic exit. Note exit routines registered with the HIS.SERVSTAT dynamic exit do not have the ability to receive instrumentation data, only monitor the state of the service. You can use the CSVDYNEX macro to control the HIS.SERVSTAT dynamic exit and its exit routines.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error.
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system will disable the exit routine after 3 consecutive abends.

Replacing the Exit Routine

For information about replacing a dynamic exit routine, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

The exit routine receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0 and must return control in same state and key with no locks held.
- In AMODE as defined by linkedit of exit routine module and RMODE=ABOVE.
- In the HIS address space as H=P=S for any of the events, or in *MASTER* (ASID 1) with H=P=S for the "service has been disabled" call only.
- With no locks held in TCB mode.

Exit Recovery

The exit routine should provide its own recovery. If the exit routine abnormally terminates, its recovery routine will get control.

If the exit routine abnormally terminates, and the exit routine does not provide its own recovery, or the error percolates beyond the exit's recovery routine, a system recovery routine will get control. If this occurs 3 consecutive times, the exit routine will be disabled.

Exit Routine Processing

The system invokes the HIS.SERVSTAT exit routine or routines, if they are registered to the dynamic exits facility, anytime the HISSERV service has enabled, started, stopped, or has disabled.

Using the Information in the Parameter List:The system passes the address of a list of parameters to the exit routine. The parameters contain the following information:

- Version of this parameter area.
- The function code, which indicates why the exit routine was called. For this exit, it would be that the status of the service has changed.
- The reason code describing how the HISSERV service has changed.
  - (1) - The service has been enabled.
  - (2) - The service has been disabled.
  - (3) - The service has started profiling the system.
  - (4) - The service has stopped profiling the system.

See macro HISYEXIT in *z/OS MVS Data Areas, Vol 2* for more details on the interface.

Programming Considerations

Observe the following conventions when coding the HIS.SERVSTAT Exit routine:

- The exit routine is allowed to obtain and release any locks it desires in order to handle the exit, however performance degradation should be a concern.

- Code the exit routine to be reentrant.
- The exit is called whenever an event begins or ends. Therefore, when coding the exit routine, you should be aware that an increased path length will increase processor utilization and may degrade performance.
- Make sure that the exit routine does not get affected when new function code support is added.

Macro Instructions and Restrictions: The exit is not allowed to issue a HISSERV REQUEST=PROFILE request, or wait on a resource held by another work unit which might issue a HISSERV REQUEST=PROFILE request.

Be aware that the exit can create performance degradation. Do not perform unnecessary tasks.

Do not code the exit routine to invoke dynamic allocation.

Entry Specifications

The system passes the address of the exit parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the exit parameter list
2-12	Not applicable
13	Address to a 216 byte save area
14	Return address
15	Entry point address of the exit routine

Parameter Descriptions: Register 1 contains the address of the exit parameter list, which is mapped by macro HISYEXIT in *z/OS MVS Data Areas, Vol 2*.

Return Specifications

The interface does not provide any field for a response; the exit does not need to return a valid value.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-1	Does not need to be preserved
2-13	Restored to contents at entry
14-15	Does not need to be preserved

Coded Example of the Exit Routine

There is no coded example of this exit routine in SYS1.SAMPLIB.

Chapter 10. ICHRTX00 — MVS Router Exit

Topics for This Exit Appear as Follows:

The system authorization facility (SAF) provides an installation with centralized control over system security processing through a system service called the MVS router. The MVS router provides a focal point for all products that provide resource management. The resource management components and subsystems call the MVS router as part of security decision-making functions in their processing, such as access control checking and authorization-related checking. These functions are called "control points". SAF supports the use of common control points across products and across systems.

To use the MVS router, a resource management component or subsystem issues the RACROUTE macro. The RACROUTE macro accepts all valid parameters for any of the independent RACF system macros (RACDEF, RACINIT, RACHECK, RACLIST, RACXTRT, and FRACHECK). RACROUTE verifies that only valid parameters have been coded and then passes the parameters to the MVS router.

For more information on the RACROUTE macro and programming requirements for the ICHRTX00 exit, see *z/OS Security Server RACROUTE Macro Reference*.

The RACROUTE macro invokes the MVS router. When it is invoked, the MVS router first calls an optional installation exit routine. If an external security product (such as RACF) is active and installed on the system, the MVS router calls it next. This process is shown in Figure 3 on page 60.

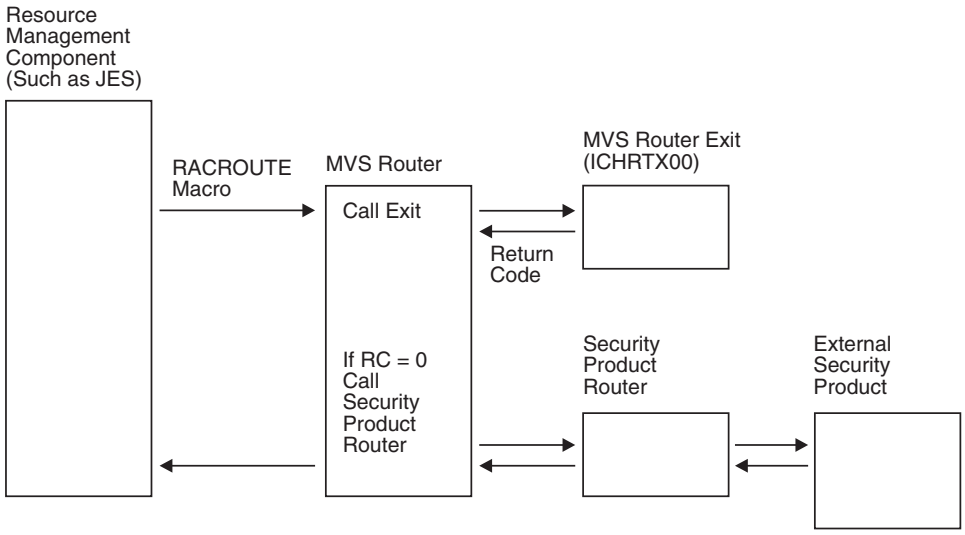

Figure 3. RACROUTE macro invokes the MVS router

If an external security product is not available, you can use the MVS router exit as an installation-written security processing (or routing) routine. If an external security product is available, you can use the MVS router exit as a preprocessing exit routine for the security product. The MVS router exit routine is ICHRTX00.

After MVS system initialization is complete, ICHRTX00 receives control for all subsequent requests for the duration of the IPL. See "Programming Considerations" on page 61 for information on coding ICHRTX00.

Installing the Exit Routine

To install ICHRTX00, name the exit ICHRTX00 and load it into the link pack area (LPA). For general instructions on installing an exit routine, see "Link editing an Installation Exit Routine into a Library" on page 3.

Exit Routine Environment

ICHRTX00 receives control in the following environment:

- Is entered via a branch and link macro. Therefore, the exit routine runs in the same key and state as the issuer of the RACROUTE macro.
- Enabled for interrupts.
- Must be linkedited with AMODE(ANY) and RMODE(24).
- Can be invoked with the local lock held.
- Caller's address space.
- Can be invoked in SRB mode. If the routine is invoked in SRB mode, the exit routine must follow SRB conventions.
- Can be invoked in cross-memory mode. If the routine is invoked in cross-memory mode, system services that invoke SVC routines cannot be used.

Exit Recovery: An installation must provide its own recovery routine for ICHRTX00. If the exit routine terminates abnormally, the recovery routine will get control first.

Exit Routine Processing

Normally, a caller invokes the MVS router and passes it class, requestor, and subsystem parameters via the RACROUTE parameter list. Using those parameters, the MVS router invokes ICHRTX00. ICHRTX00 returns to the MVS router with a return code that indicates whether further security processing is to occur.

If the return code is 0, the MVS router invokes the external security product by calling its router, ICHRFR00. ICHRFR00 will then invoke the other external security product processing and will report the results of that invocation to the MVS router by placing a return code in register 15 and the detailed RACF-compatible return and reason codes in the first and second words (respectively) of the RACROUTE parameter list. For more information on the return codes the exit routine can set, see the description of registers at exit in "Return Specifications" on page 62.

Simulating a Call to RACF: Instead of invoking the external security product, your installation may choose to have ICHRTX00 respond to the caller's request. In that case, you must still provide the caller with the RACF-compatible return and reason codes that it expects to receive. To do so, set the exit routine return code so that the external security product is not invoked (as described in "Return Specifications" on page 62). However, you must simulate the results of an external security product invocation by coding ICHRTX00 so it places the RACF-compatible return and reason codes in the RACROUTE parameter list.

RACF return and reason codes are documented in *z/OS Security Server RACROUTE Macro Reference*.

Programming Considerations

ICHRTX00 must be reentrant.

In addition to the address of the RACROUTE parameter list, ICHRTX00 also receives the address of a 152-byte work area.

SAF performs functions other than being a router, such as creating security tokens for certain RACROUTE request types, propagating userids, and creating default control blocks (ACEEs) when an external security product is not available to the system. IBM recommends that, in coding ICHRTX00, you do not bypass these SAF functions. SAF creates and returns control blocks (tokens or ACEEs) whenever the RACROUTE request types are issued:

- REQUEST=VERIFYX
- REQUEST=TOKENMAP
- REQUEST=TOKENXTR
- REQUEST=TOKENBLD

SAF also creates default ACEEs for REQUEST=VERIFY when an external security product is not available on the system. System code, such as JES, requires these control blocks. Therefore, if your ICHRTX00 exit routine bypasses SAF security functions, your installation must construct and return the control blocks that SAF would have created. If you do not provide the required control blocks, problems can result. The token fields are mapped by macro ICHRUTKN (data area RUTKN). For a mapping of the RUTKN data area, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Macro Instructions and Restrictions: Do not install an exit routine that issues the WAIT macro or calls a service that issues a WAIT, such as WTOR. WAITs and implied WAITs can cause the system console or JES to stop functioning.

Entry Specifications

The MVS router passes to the exit routine (in Register 1), the address of a doubleword area that contains the addresses of:

- The RACROUTE parameter list and
- A work area that the exit routine can use.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register
Contents

0 Not applicable

1 Address of the following area:

Offset	Length	Description
0	4	**Parameter list address:** points to the RACROUTE parameter list
4	4	**Work area address:** points to a 152-byte work area that ICHRTX00 can use.

2-12 Not applicable

13 Register save area

14 Return address

15 Entry point address of the exit

Parameter Descriptions: The RACROUTE parameter list (SAFP) is mapped by macro ICHSAFP (data area SAFP). If an ICHRTX00 exit routine exists, the MVS router passes the SAFP to the exit. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for a mapping of the SAFP data area.

Return Specifications

A return code from the exit routine indicates whether the external security product is to be given control or further security processing is to be bypassed.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-14 Restored to contents at entry.

15 One of the following return codes:

Hex/Dec
Explanation

X'0' (0)
The exit has completed successfully. Control proceeds to the external security product router (ICHRFR00) for further security processing and an invocation of the external security product.

X'C8' (200)
The exit routine has completed successfully. The MVS router translates this return code to an MVS router return code of 0 and returns control to the issuer of the RACROUTE macro, bypassing further SAF and any external security product processing. (See note.)

X'CC' (204)
The exit routine has completed successfully. The MVS router translates this return code to an MVS router return code of 4 and returns control to the issuer of the RACROUTE macro, bypassing further SAF and any external security product processing. (See note.)

X'D0' (208)
The exit routine has completed processing. The MVS router translates this return code to an MVS router return code of 8 and returns control to the issuer of the RACROUTE macro, bypassing further SAF and any external security product processing. (See note.)

Other If the exit routine sets any other return code, the MVS router returns control directly to the issuer of the RACROUTE macro and passes the untranslated code as the MVS router return code. Further SAF and any external security product processing is bypassed.

Note: The installation is responsible for putting RACF-compatible return and reason codes in the first 2 fullwords, respectively, of the RACROUTE parameter list (SAFP). If the exit routine does not issue a specific reason code, it should issue a zero reason code.

Coded Example of the Exit Routine

A copy of a sample ICHRTX00 exit routine is provided in SYS1.SAMPLIB (in member RACINSTL).

Chapter 11. IEALIMIT — User Region Size Limit Exit

Topics for This Exit Appear as Follows:

An installation can limit application programs' access to nonextended private area storage (subpools 0-127, 129-132, 244, 251 and 252) by writing IEALIMIT. This exit routine is invoked before each job step is started. IEALIMIT can be used under MVS to set nonextended region size and nonextended region limit only. The values set by IEALIMIT should be less than the size of the nonextended private area. If they are not, the control program uses the size of the nonextended private area.

The values set by IEALIMIT have no effect on establishing the extended region size and extended region limit. It is recommended that the exit routine IEFUSI be used in place of IEALIMIT when possible, especially when jobs are expected to specify a region value greater than 16 megabytes. IEFUSI can include region limit processing for the private area for both less than and greater than 16 megabytes. See Chapter 32, "IEFUSI — Step Initiation Exit," on page 211 for a comparison of these two exit routines.

Limiting Region Size Consideration: If you want to use the IEALIMIT exit to control region size, and you have the IEFUSI exit available, ensure that the flag to bypass the IEALIMIT exit is not set in the IEFUSI parameter list.

Installing the Exit Routine

The IBM-supplied IEALIMIT exit routine is linkedited with the nucleus. To replace the IBM-supplied routine with your own IEALIMIT, you must linkedit your own version into the nucleus prior to an initial program load.

For general instructions on installing an exit routine, see "Link editing an Installation Exit Routine into a Library" on page 3.

Exit Routine Environment

IEALIMIT receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.

- In AMODE 24 and RMODE 24.
- In any address space.
- Holds a local lock.
- Under the initiator's task.

Exit Recovery: The exit routine runs under an ESTAE. The VSM recovery routine that gets control should an error occur in the IEALIMIT exit will (1) fill in the SDWA (with IEALIMIT as the module name) and take an SVC dump, and then (2) either (a) retry the operation (including another call to the IEALIMIT exit) or (b) percolate, which could eventually result in an ABEND.

Because IEALIMIT is an authorized routine that runs in key 0, it can provide its own recovery routine, which would have the same capabilities as the IEALIMIT routine itself. The recovery routine would be subject to the limitations on all recovery routines, which are documented in the chapter "Providing Recovery" of *z/OS MVS Programming: Authorized Assembler Services Guide*.

Exit Routine Processing

IEALIMIT gets control after an existing region has been freed, and before a new region is initialized. On entry, the IEALIMIT routine receives in register 0 the number of bytes requested by the application program for its region, as specified through the JCL REGION parameter. If this REGION value is less than 16 megabytes, the value in register 0 is the same as the REGION value. If this REGION value is greater than 16 megabytes, the value in register 0 is equal to the size of the nonextended private area minus 64K. Register 1 contains the same value as register 0.

If the JCL REGION parameter value is zero or if the parameter is absent, then the installation JES default value is used. If the JES default is zero, the IEALIMIT routine receives a zero in register 1.

After processing, the IEALIMIT routine returns values in both registers 0 and 1. Register 0 contains the number of bytes to be used as the region size which should be less than the size of the nonextended private area. Register 1 contains the number of bytes to be used as the region limit. Both registers 0 and 1 should be rounded to a multiple of 4K. These values in registers 0 and 1 determine how much space is allocated in the user's region, in response to GETMAIN and STORAGE requests.

IEALIMIT Default Values: If your installation does not supply an IEFUSI exit routine to control region size, and does not override the default values in the IBM-supplied IEALIMIT exit routine, users obtain the results shown in Table 1 when specifying various values for the region size on the REGION parameter.

Table 1. IEALIMIT Default Values

JCL Region parameter	Results-below 16 Mb	Results-above 16 Mb
0K or 0M	The job step is allocated all the storage available below 16 megabytes. The resulting size of the region below 16 megabytes is 16 megabytes minus the amount of virtual storage allocated to MVS.	The job step is allocated all the storage available above 16 megabytes. The resulting size of the region above 16 megabytes is 2 gigabytes minus the amount of virtual storage allocated to MVS, minus 16 megabytes.

Table 1. IEALIMIT Default Values (continued)

JCL Region parameter	Results-below 16 Mb	Results-above 16 Mb
> 0K or 0M and <= 16384K or 16M	Establishes the size of the private area below 16 megabytes. If the region size specified is not available below 16 megabytes, the job (or job step, if coded on the EXEC statement) abnormally terminates with an ABEND 822.	The extended region size is the default value of 32 megabytes
> 16384K or 16M and <= 32768K or 32M	The job (or job step) is allocated all the storage available below 16 megabytes. The resulting size of the region below 16 megabytes is 16 megabytes minus the amount of virtual storage allocated to MVS.	The extended region size is the default value of 32 megabytes.
> 32768K or 32M and <= 2096128K or 2047M	The job (or job step) is allocated all the storage available below 16 megabytes. The resulting size of the region below 16 megabytes is 16 megabytes minus the amount of virtual storage allocated to MVS.	The extended region size is the specified value. If the region specified is not available above 16 megabytes, the job step receives whatever storage that is available above 16 megabytes, up to the requested amount. The resulting size of the region above 16 megabytes depends on system options and on what system software is installed.

See *z/OS MVS Initialization and Tuning Guide* for further information on the effect of the region limit and region size on requests for storage.

Programming Considerations

Code IEALIMIT to be reentrant. If this exit routine uses dynamic storage, use subpool 229, 230, or 249 for that storage. Do not use 0-127, as this will determine the key of the subpool for the duration of the jobstep. Common storage and LSQA are not recommended.

When using IEALIMIT to limit region size, the region size should be less than the region limit. This provides protection against programs that issue variable length GETMAINs with very large maximums and then do not immediately free part of that space, or free such a small amount that a subsequent GETMAIN (possibly issued by a system service) fails.

Entry Specifications

Either the value specified by the REGION parameter on the JOB or EXEC statement or the value of the JES default is passed to the IEALIMIT routine so that the exit can determine whether it is acceptable.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register
Contents

0 If the application program requests a region that is less than 16 megabytes, then register 0 contains the size of the region requested by the application program.

If the application program requests a region that is greater than 16 megabytes, then register 0 contains a value equal to the size of the available private area minus 64K.

Note: An application program explicitly requests a region that is less than 16 megabytes by specifying 'REGION=xxM' on the JOB or EXEC JCL statements and making 'xx' less than or equal to 16.

An application program explicitly requests a region that is greater than 16 megabytes by specifying 'REGION=xxM' on the JOB or EXEC JCL statements and making 'xx' larger than 16.

An application program requests a default region by omitting the REGION parameter from both the JOB and EXEC JCL statements. The value that is passed in register 0 is controlled by JES.

1 The contents are the same as register 0.

2-12 Not applicable

13 Register save area

14 Return address

15 Entry point address of IEALIMIT

Return Specifications

The IEALIMIT routine returns a region size and region limit.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0 The number of bytes to be used as the region size. This number should be less than the value in register 1.

1 The number of bytes to be used as the region limit. This value should be less than the size of the nonextended private area.

2-15 Restored to contents at entry

Coded Example of the Exit Routine

Figure 4 is a coded example of the IEALIMIT exit routine.

```
* Save caller's registers and establish addressability.
          STM   14,15,12(13)
          STM   2,12,28(13)
          BALR  9,0
* If this is not a request for an unlimited
* region, then increase the region limit by 64K.
* The region size is not changed.
          LTR   1,1
          BZ    EXIT
          AL    1,INCRMENT
* Restore caller's registers and return to caller.
EXIT      LM    14,15,12(13)
          LM    2,12,28(13)
          BR    14
INCRMENT  DC    F'65536'
```

Figure 4. Example: IEALIMIT exit routine

The IBM-supplied IEALIMIT routine (above) does the following processing:

- If register 1 contains a nonzero value, the IEALIMIT routine adds 64K to its contents and returns to the caller. This value in register 1 is used to limit the allocation of storage from subpools 0-127, 129-132, 244, 251, and 252. The content of register 0 is unchanged.
- If register 1 contains a zero, the IEALIMIT routine returns a zero in register 1 to the caller. This indicates that no limit is assigned to the job, started program, or TSO/E user.

Chapter 12. IEAVADFM — Format SNAP, SYSABEND, and SYSUDUMP Dumps

Topics for This Exit Appear as Follows:

IEAVADFM is a dump facility installation exit routine name list. It contains a list of installation exit routine names to be given control during the formatting of a SNAP or ABEND dump. An installation can use these exit routines to:

- Gather information to be included in a SNAP/ABEND dump and
- Format the information to be written to a data set described by a SYSABEND, SYSUDUMP, or installation-defined JCL DD statement.

The installation exit routines listed in IEAVADFM are invoked during the control block formatting phase of every SNAP or ABEND dump for which the CB option was specified. The system provides to the exit routines:

- A buffer in which the routines can build a print line
- The address of an IBM-supplied print routine to which the exit routines can pass the line for printing.

The difference between IEAVADFM and IEAVADUS is that IEAVADFM is an ***installation exit name list*** while IEAVADUS is a ***single exit routine***. The routine names listed in IEAVADFM are invoked sequentially and maintain control until the end of the list of routines is reached or until a routine within the list returns a terminating code. IEAVADUS receives control once for each SNAP dump or ABEND dump and then returns control to the calling program.

Installing the Exit Routine

To install a dump formatting installation exit routine in your system, you need to do the following :

- Linkedit the dump formatting exit routine into SYS1.LPALIB, SYS1.LINKLIB, or a data set in the LNKLST concatenation.
- Define the dump formatting exit routine to MVS by adding its load module name to the exit name list in IEAVADFM.

Defining Dump Formatting Exits to IEAVADFM

You can specify dump formatting exits in any of the blank entries in the IEAVADFM exit name list. During the formatting of a SNAP or ABEND dump, the system invokes the routines in the order you specified them. That is, the exit routine specified in entry 1 gets control first, followed by the exit routine specified in entry 2, and so on.

IEAVADFM is a CSECT in load module IGC0805A (which resides in SYS1.LPALIB). The IBM-supplied version of IEAVADFM contains the following entries:

- Four 8-byte entries that contain hexadecimal zeroes, and are intended for customer use, followed by
- A final 4-byte entry that contains hexadecimal zeroes to indicate the end of the exit name list (the end-of-table marker).

You can add as many dump formatting exit routine load modules to IEAVADFM as you want. IBM recommends that you use the existing blank entries in IEAVADFM before creating new ones. Ensure that the last entry contains 4 bytes of hexadecimal zeroes.

Contents of Entries: The first four 8-byte entries contain the exit load module name. These entries contain hexadecimal zeroes, and are available for customer use.

This is how the first, second, and last entries in IEAVADFM appear:

```
IEAVADFM
HEX LOCATION:     CONTENTS:
0000              00000000 00000000       ENTRY 1
0008              00000000 00000000       ENTRY 2
 .                        .                  .
 .                        .                  .
 .                        .                  .
0040              00000000                END OF LIST
```

To insert a dump formatting exit load module name in the list, select an available entry (one that is set to hexadecimal zeroes) and substitute the exit load module name in place of the hexadecimal zeroes. Exit names can be one to eight characters, padded to the right with blanks.

Examples of changing entries in IEAVADFM are shown in the sections that follow .

Adding and Deleting Exit Names in IEAVADFM: To add or delete exit names in the exit name list, you must modify the IEAVADFM object code in SYS1.LPALIB through the use of the SPZAP program, or through SMP/E. For more information, refer to “Using SPZAP or SMP/E to Add or Delete Name.”

Using SPZAP or SMP/E to Add or Delete Name

Adding Exit Names: When using SPZAP or SMP/E, take the following steps:

- Use SPZAP to produce a dump of IEAVADFM. Sample JCL follows:

```
//DUMPJCL  JOB MSGLEVEL=(1,1)
//STEP     EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSNAME=SYS1.LPALIB,DISP=OLD
```

```
//SYSIN    DD *
 NAME IEAVADFM
 DUMP IEAVADFM
/*
```

- Use the dump produced to select an available entry in IEAVADFM.
- Write the name of the dump formatting exit module in the entry. (See Example 1 if you are using SPZAP or Example 1A if you are using SMP/E.)

The changes will take affect on the next IPL.

Example 1 - Using SPZAP to Add Exit Names: The following job adds EXITRTN1 to the first entry in IEAVADFM, and EXITRTN2 to the second entry in IEAVADFM. The job then dumps IEAVADFM to verify the changes.

```
//EXAMPLE1 JOB MSGLEVEL=(1,1)
//STEP     EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSNAME=SYS1.LPALIB,DISP=OLD
//SYSIN    DD *
 NAME IEAVADFM
 VER 0000 0000,0000,0000,0000    FIRST UNUSED ENTRY
 REP 0000 C5E7,C9E3,D9E3,D5F1    SET TO EXITRTN1
 VER 0008 0000,0000,0000,0000    SECOND UNUSED ENTRY
 REP 0008 C5E7,C9E3,D9E3,D5F2    SET TO EXITRTN2
 DUMP IEAVADFM
/*
```

Example 1A - Using SMP/E to Add Exit Names: The following job adds EXITRTN1 to the first entry in IEAVADFM, and EXITRTN2 to the second entry in IEAVADFM.

```
++USERMOD(USRMOD2).
++VER(Z038) FMID(HBB4410). /* CHANGE THE FMID AS NEEDED */
++ZAP(IEAVADFM).
 NAME IEAVADFM IEAVADFM
 VER 0000 0000,0000,0000,0000    FIRST UNUSED ENTRY
 REP 0000 C5E7,C9E3,D9E3,D5F1    SET TO EXITRTN1
 VER 0008 0000,0000,0000,0000    SECOND UNUSED ENTRY
 REP 0008 C5E7,C9E3,D9E3,D5F2    SET TO EXITRTN2
```

Deleting Exit Names: When using SPZAP or SMP/E, take the following steps:

- Use SPZAP to dump IEAVADFM (as shown earlier in the sample JCL code under "Using SPZAP or SMP/E to Add or Delete Name" on page 72).
- Write X'40' (blanks) in place of the dump formatting exit module name. (See Example 2 if you are using SPZAP or Example 2A if you are using SMP/E.)

The changes will take effect on the next IPL.

If all entries are in use, you can replace module IEAVADFM, or expand it. Ensure that the last entry contains 4 bytes of hexadecimal zeroes.

Example 2 - Using SPZAP to Delete Exit Names: The following job deletes EXITRTN1 from the dump formatting exit list. The job then dumps IEAVADFM to verify the changes.

```
//EXAMPLE2 JOB MSGLEVEL=(1,1)
//STEP     EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSNAME=SYS1.LPALIB,DISP=OLD
//SYSIN    DD *
```

```
 NAME IEAVADFM
 VER 0000 C5E7,C9E3,D9E3,D5F1    EXITRTN1
 REP 0000 4040,4040,4040,4040    RESTORE ENTRY 1
 DUMP IEAVADFM
/*
```

Example 2A - Using SMP/E to Delete Exit Names: The following job deletes EXITRTN1 from the dump formatting exit list.

```
++USERMOD(USRMOD3).
++VER(Z038) FMID(HBB4410). /* CHANGE THE FMID AS NEEDED */
++ZAP(IEAVADFM).
 NAME IEAVADFM IEAVADFM
 VER 0000 C5E7,C9E3,D9E3,D5F1    EXITRTN1
 REP 0000 4040,4040,4040,4040    REPLACE WITH BLANKS
```

For additional information on the use of:

- SPZAP, see *z/OS MVS Diagnosis: Tools and Service Aids*.
- SMP, see *SMP/E for z/OS User's Guide*.
- Expand, see *z/OS MVS Program Management: User's Guide and Reference*.

Exit Routine Environment

Each routine in IEAVADFM receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In the AMODE and RMODE specified in the routine or in the linkedit.
- In the address space of the task taking the dump.
- With no locks held.
- Under the task associated with the request block that requested the dump. Each routine runs under the SVRB of SNAP/ABDUMP.

Exit Recovery: An ESTAE routine provides recovery for SNAP. Each user formatting routine listed in IEAVADFM should, however, also set up its own recovery to handle any ABENDs encountered during the formatting process. Each recovery routine should either recover and continue, or recover and return to SNAP with a zero return code. A nonzero return code is interpreted as a GETMAIN failure, causing the following message in the dump data set:

```
USER/PP CONTROL BLOCKS UNAVAILABLE
```

The dump is truncated because of lack of storage. The recovery routine should not continue formatting if a X'37' ABEND occurs, because no space remains in the dump data set. Before the recovery routine returns to SNAP, it should free all the storage that it has obtained.

If the user formatting routine does not establish recovery, or if the recovery exit specifies continue-with-termination after an ABEND, SNAP terminates this control block formatter entirely and continues with the next portion of the dump, if any.

Exit Routine Processing

The installation exit routines listed in IEAVADFM receive control automatically during the control block formatting phase of every SNAP or ABEND dump for which the CB option was requested.

The routines build one print line at a time in the buffer provided and use BALR to branch to the IBM-supplied print routine, which in turn prints the line to the dump data set. Offsets are recommended for all formatted control blocks that are longer than one output line. (One line generally formats 20 hexadecimal characters). The print routine saves registers, prints the line, blanks the buffer, restores the registers, and returns control to the user's routine via register 14.

Programming Considerations

Code each routine in the IEAVADFM name list to be reentrant.

In order to avoid an abnormal termination later in the SNAP/ABEND routine, the user's routines must not free either the entry parameter list or the print buffer.

IEAVADFM works through the IBM-supplied print routine, so the formatting exit does not have any direct access to the carriage controls. Therefore, in order to cause a skipped line in the dump output, you must pass a blank buffer to the print routine. The print routine handles page ejects. The installation exit routine can use format patterns to format data in the output buffer, by using the IBM-supplied format service routine. The service routine can also convert data to printable hexadecimal. This service routine is the same routine that is provided by the IPCS service aid; see *z/OS MVS IPCS Customization*. No registers are necessary as input to this service routine.

IBM-Supplied Print Routine: The installation exit routines listed in IEAVADFM can use the IBM-supplied print routine to format data to the dump data set. The print routine is pointed to by the ADPLPRNT field in the exit parameter list and is mapped by macro BLSABDPL (data area BLSABDPL). For a mapping of the BLSABDPL data area, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

The interface to the print routine is:

- *Entry:* via BALR 14,15 for each line to be written.
- *Environment:* PSW key 0, supervisor state, no locks held.

Entry Specifications

The ABDUMP calling routine passes to each installation exit routine in IEAVADFM the address of a parameter list useful for formatting the dump data.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the parameter list (mapped by IHAABDPL) for the user formatting routine.
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the installation exit routine listed in IEAVADFM

Parameter List Contents: Register 1 points to a parameter list that provides the addresses of subroutines and data that the user-written format routines will use. This is the same parameter list used by exit routine IEAVADUS.

This parameter list is mapped by the BLSABDPL mapping macro (data area BLSABDPL). The mapping list includes all the fields of the IPCS service aid's parameter list so user formatting routines can be invoked by either SNAP/ABEND or IPCS.

The BLSABDPL mapping is documented in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

The installation exit routine returns a code indicating whether processing should continue.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents at entry
15	One of the following return codes:

Return Code	Explanation
0	Continue processing.
4	Request is not valid.
12	Suppress the remainder of the SNAP/ABEND dump. If the same installation exit routine is executed under the IPCS service aid, print dump does not suppress the remainder of the dump.

Chapter 13. IEAVADUS — Select and Format Dump Data Exit

Topics for This Exit Appear as Follows:

You can use the IEAVADUS installation exit to select and format data to be included in an ABDUMP (SNAP/ABEND dump). The selected data is written to a data set described by a SYSABEND, SYSUDUMP, or installation-defined JCL DD statement. The system provides an area in which IEAVADUS builds a print line and also provides the address of an IBM-supplied print routine to which the installation exit routine passes the line for printing.

The difference between IEAVADUS and IEAVADFM is that the exit routine IEAVADUS is a ***single exit routine*** while IEAVADFM is an ***installation exit name list***. IEAVADUS receives control once for each SNAP dump or ABEND dump. Your IEAVADUS installation exit routine replaces one already supplied with your system. In contrast, the routine names in IEAVADFM are invoked sequentially when IEAVADFM receives control. IEAVADFM maintains control until the end of the list of routines is reached or until a routine within the list returns a terminating code.

Installing the Exit Routine

To use IEAVADUS, you must linkedit the exit routine into SYS1.LPALIB with the load module name of IGC0905A, replacing the IBM-supplied routine IEAVADUS.

To remove the installation exit routine from the system, linkedit a copy of module IEFBR14 into SYS1.LPALIB with the name IGC0905A.

Exit Routine Environment

IEAVADUS receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In the AMODE and RMODE specified in the routine or in the linkedit.
- In the address space of the task taking the dump.
- With no locks held.

- Under the task associated with the request block that requested the dump. The routine runs under the SVRB of SNAP/ABEND.

Exit Recovery: An ESTAE routine provides recovery for SNAP. IEAVADUS should, however, also set up its own recovery to handle any ABENDs encountered during the formatting process. The routine should either recover and continue, or recover and return to SNAP with a zero return code. A nonzero return code is interpreted as a GETMAIN failure, causing the following message in the dump data set:

```
USER/PP CONTROL BLOCKS UNAVAILABLE
```

The dump is truncated because of lack of storage. The recovery routine should not continue formatting if a X'37' ABEND occurs, because no space remains in the dump data set. Before the recovery routine returns to SNAP, it should free all the storage that it has obtained.

If the IEAVADUS routine does not establish recovery, or if the recovery exit specifies continue-with-termination after an ABEND, SNAP terminates this control block formatter entirely and continues with the next portion of the dump, if any.

Exit Routine Processing

IEAVADUS receives control automatically during the control block formatting phase of every SNAP and ABEND dump for which the CB option was requested.

IEAVADUS builds one print line at a time in the buffer whose address is in the exit parameter list (in the ADPLBUF field). To print the line on the dump data set, IEAVADUS invokes the IBM-supplied print routine (via BALR or CALL). Offsets are recommended for all formatted control blocks that are longer than one output line. (One line generally formats 20 hexadecimal characters.) The print routine saves registers, prints the line, blanks the buffer, restores the registers, and returns control to IEAVADUS via register 14.

When all lines have been selected and printed, IEAVADUS restores the entry registers and returns to SNAP/ABDUMP.

Note: IEAVADUS can format and print an entire control block in one invocation of the IPCS control block formatter service. See “Writing IPCS Exit Routines” in *z/OS MVS IPCS Customization* for information on how to define a control block model to SNAP/ABDUMP.

Programming Considerations

Code your IEAVADUS routine to be reentrant.

Return from IEAVADUS to SNAP/ABDUMP must be made in protection key 0, supervisor state, with no locks held (the same state as when IEAVADUS was entered).

In order to avoid an abnormal termination later in the SNAP/ABEND routine, the user's routines must not free either the entry parameter list or the print buffer.

Because IEAVADUS works through the IBM-supplied print routine, IEAVADUS has no direct access to the carriage controls. Therefore, to cause a skipped line in the dump output, you must code IEAVADUS to pass a blank buffer to the print routine.

The print routine handles page ejects.

IEAVADUS can use format patterns to format data in the output buffer by using the IBM-supplied format service routine. The service routine can also convert data to printable hexadecimal. This service routine is the same routine that is provided by the IPCS service aid; see *z/OS MVS IPCS Customization*.

IEAVADUS can use the subpool indicated in ADPLSBPL for all working storage.

The IPCS Control Block Formatter Service: Before invoking the IPCS control block formatter service, IEAVADUS initializes several fields in the exit parameter list (in the ADPLPFMT area) as follows:

Field IEAVADUS Sets to:

ADPLPBLC
Number of blank lines the formatter will skip

ADPLPCHA
Control block acronym (for example, "TCBnnnn")

ADPLBAS
0

ADPLPBLS
0

ADPLPBAV
The virtual address of the control block

ADPLPVCL
The view control; IEAVADUS sets this field to select the individual fields to be printed in the dump

Invoking the Service: Code IEAVADUS to take the following steps to invoke the IPCS control block formatter service:

1. Create and pass a 3-word parameter list to the control block formatter routine, ADPLESRV. Before passing the 3-word parameter list, IEAVADUS places the values shown in Table 2 in the parameter list.

Table 2. Values Passed to ADPLESRV

Field	Description
Word 1	Address of the ABDPL
Word 2	Address of the control block service code (ADPLSCBF)
Word 3	Address of the control block formatter service parameter list (ADPLPFMT)

2. Set register to the address of the 3-word parameter list
3. Call the exit services router (its address is in the ADPLSERV field of the exit parameter list). The remaining fields needed by the control block formatter service have already been initialized by SNAP/ABDUMP.

See "Using the IBM-supplied Exit Service Routines" in *z/OS MVS IPCS Customization* for more information about the IPCS control block formatter service.

The IBM-Supplied Print Routine: IEAVADUS can use the IBM-supplied print routine to format data to the dump data set. The print routine is pointed to by the ADPLPRNT field in the exit parameter list and is mapped by macro BLSABDPL

(data area BLSABDPL). For a mapping of the BLSABDPL data area, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

The interface to the print routine is:

- *Entry:* via BALR 14, 15 for each line to be written.
- *Environment:* PSW key 0, supervisor state, no locks held.

Entry Specifications

The ABDUMP calling routine passes to IEAVADUS the address of a parameter list that contains information the exit routine can use to format the dump data.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of a parameter list (mapped by IHAABDPL) for the user formatting routine
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEAVADUS

Parameter List Contents: Register 1 points to a parameter list that provides the addresses of subroutines and data that the user written format routines will use.

This parameter list is mapped by the BLSABDPL mapping macro (data area BLSABDPL). The mapping list includes all the fields of the IPCS service aid's parameter list so user formatting routines can be invoked by either SNAP/ABEND or IPCS.

The BLSABDPL mapping is documented in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

A return code from IEAVADUS indicates whether the exit routine is to continue processing or to suppress the remainder of the dump.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents at entry
15	One of the following return codes:

Return Code	Explanation
0	Continue processing.

4 Request is not valid.

12 Suppress the remainder of the ABEND dump. If the same exit routine is executed under the IPCS service aid, print dump does not suppress the remainder of the dump.

Chapter 14. IEAVMXIT — Installation-Specified MPF Exits

Topics for This Exit Appear as Follows:

The IEAVMXIT installation exit or an MPF installation exit (one that you specify on the USEREXIT parameter in an MPFLSTxx member of SYS1.PARMLIB) allows you to modify message processing in a system or sysplex. IEAVMXIT is the general-purpose exit routine that does processing that is common to many messages (WTOs). An MPF exit routine does processing that is specific to a certain type of message or a particular message ID.

For information on the MPFLSTxx member of SYS1.PARMLIB, see *z/OS MVS Initialization and Tuning Reference*.

You can use IEAVMXIT or an MPF exit routine to:

- Modify the presentation of messages by:
  - Changing the text and descriptor codes of selected messages.

    Changing the descriptor code can alter the retention of the message on a console screen and in the Action Message Retention Facility (AMRF). It can also affect the color of a message when it is displayed on a console with color capabilities.
  - Changing the color, intensity, and highlighting of messages.
- Modify the routing of messages by:
  - Changing the routing codes of selected messages.
  - Changing either the console name or the console ID to which the message is queued.
  - Selectively routing messages to a specific console.
  - Queuing messages to a particular active console.
  - Queuing messages by routing codes.
  - Directing messages to hardcopy only.

  - Indicating whether or not to broadcast a message to active consoles.
  - Reducing message traffic at specific consoles by redirecting some traffic.
  - Indicating whether or not the message should be routed to the consoles that requested to see the message as the result of a MONITOR command.
- Reduce operator workload through message suppression or automation by:
  - *Selectively* suppressing (filtering) occurrences of messages. (MPF suppresses *all* occurrences of a particular message.)
  - Performing error thresholding.
  - Indicating whether or not the action message retention facility (AMRF) is to retain an action message.
  - Overriding message processing facility (MPF) suppression.
  - Handling the common requests (WTORs) from the system.
  - Altering the automation token specified in the MFPLSTxx member of SYS1.PARMLIB.
  - Indicating whether to consider a message for automation.
  - Deleting a message

If an IEAVMXIT message exit already exists in the installation and you require that the installation uses message flood automation, you can either integrate the existing IEAVMXIT with message flood automation, or call message flood automation from IEAVMXIT. For more details about customizing the IEAVMXIT, see *z/OS MVS Planning: Operations*.

Installing the Exit Routine

IEAVMXIT: The IEAVMXIT exit routine is an installation-coded module. When you install this exit routine you must name it IEAVMXIT.

Specify whether you want to have IEAVMXIT active or not active at IPL by specifying either (Y) or (N) on the UEXIT keyword on the INIT statement of the CONSOLxx parmlib member. If you do not specify the UEXIT keyword, the system assumes the default, which is UEXIT(Y), and activates IEAVMXIT if it is installed.

You must provide your own IEAVMXIT routine if you specify UEXIT with the (Y) option or expect the system to default to (Y).

If your IEAVMXIT routine is active at IPL, it will be invoked for all of the messages that were issued during IPL and NIP. This invocation is done after NIP is over, when the messages are re-issued to be recorded in the hardcopy log.

Operators can use the CONTROL M command to change the online status of IEAVMXIT.

You can insert your IEAVMXIT exit routine into the control program by:

- Linkediting it into the LNKLST. Use 31-bit addresses in the routine and assemble it with AMODE 31 and RMODE ANY.
- Activating it with the CONTROL M, UEXIT=Y command.

MPF Exit Routine: When an MPF exit routine is installed, its address is located during the processing of the SET MPF command (and the associated processing of

the specified MPFLSTxx member of SYS1.PARMLIB). When the MPF exit routine is to be invoked, its address is passed to the installation exit interface, and the exit routine is invoked via standard linkage.

Do not use the name IEAVMXIT as the name of an MPF exit that you specify in the MPFLSTxx parmlib member.

Operators can use the SET MPF command to change the online status of MPF exit routines.

You can insert MPF exit routines into the control program by following these steps:

1. Link-edit them into an APF-authorized library that is part of the LNKLST concatenation. Use 31-bit addresses in the routines and assemble them with AMODE 31 and RMODE ANY.
2. Put the name of each MPF exit routine you write into the MPFLSTxx parmlib member. Specify the name of the exit routine on the USEREXIT parameter of the message ID entry for each message the exit routine is to process.
3. Activate the MPFLSTxx member with a SET MPF=xx command.

Replacing the Exit Routine Without a Re-IPL:

There may be times when you need to replace IEAVMXIT or an MPF exit routine, either because you want to add functions to the routine or because the routine abended when it was processing a particular message. Depending on whether the routine is IEAVMXIT or an installation-specified MPF exit routine, the procedures are as follows:

IEAVMXIT: To replace your IEAVMXIT exit routine with a fresh copy, take the following steps:

- Linkedit the new copy of IEAVMXIT into the LNKLST.
- Refresh LLA with the MODIFY LLA,REFRESH command.

  If you make the message exit available by changing the libraries referred to in the LINKLIST concatenation, you must issue the

  ```
  SETPROG LNKLST, UPDATE, JOB=CONSOLE
  ```

  command to cause the Console address space to use the new LINKLIST concatenation. The K M command runs in the Console address space.
- Reload and reactivate the exit routine using the K M, UEXIT=Y command.

MPF Exit Routine: To replace an MPF exit routine with a fresh copy, take the following steps:

- Link-edit the new copy of the exit routine into an APF-authorized library that is part of the LNKLST concatenation.
- Refresh LLA with the MODIFY LLA,REFRESH command.

  If you make the message exit available by changing the libraries referred to in the LINKLIST concatenation, you must issue the

  ```
  SETPROG LNKLST, UPDATE, JOB=*MASTER*
  ```

  command to cause the Master Schedule address space to use the new LINKLIST concatenation. The SET MPF command runs in the Master Schedule address space.
- Reload and reactivate the exit routine using the SET MPF=xx command.

Exit Routine Environment

The IEAVMXIT and MPF exit routines receive control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31 and RMODE ANY.
- With no locks held; they MUST return control with no locks held.
- Address space of the WTO issuer.

Exit Recovery: The IEAVMXIT and MPF exit routines must provide their own level of recovery because, with one exception, the system does not continue to pass control to an exit routine after it abnormally terminates.

The exception is when an exit routine is to be deactivated (via the CONTROL M command for IEAVMXIT, and the SET MPF command for MPF exit routines) and the contents of the exit routine's individual data area are nonzero. In this case, the routine is given control before it is deactivated, so that it can clean up any work areas it may have created.

For information on how to reactivate the exit routine if it abnormally terminates, see “Replacing the Exit Routine Without a Re-IPL:” on page 85.

For information on the individual data area, see “Programming Considerations” on page 88.

Exit Routine Processing

The IEAVMXIT and MPF exit routines get control during MPF processing. The IEAVMXIT and MPF exit routines are mutually exclusive. For a particular message ID, if you have not named an MPF exit routine to do specific processing, IEAVMXIT, the general-purpose exit routine, receives control.

The control program calls IEAVMXIT or an MPF exit routine for all single-line messages. For a multiple-line message, the program calls the exit routine only for the first line of the message, unless the routine requests minor-line processing. When the exit routine requests minor-line processing, all minor lines will be processed. The default is to bypass minor-line processing.

Message Processing Considerations: Your exit will receive all of the parameters of the message in a parameter list that is known as the CTXT. The CTXT is mapped by macro IEZVX100 that is found in the SYS1.MODGEN system library. When planning to write the IEAVMXIT or an MPF exit routine, you should consider carefully the steps that are necessary to process the message. One step might be sufficient to obtain your desired result; other cases might require several steps.

Request Processing: Your exit can examine all of the attributes of a message and can alter almost all of them. To alter an attribute of a message, you must alter the attribute in the CTXT parameter list and indicate through a "request flag" that the alteration is to be made in the actual message. You can only alter fields that have request flags associated with them. Alterations to fields that do not have request flags associated with them will be ignored.

Although you can alter many of the attributes of a message, you cannot convert a single-line message into a multi-line message or a multi-line message into a

single-line message. You cannot convert a single-line message into a Write To Operator with Reply (WTOR) message or a WTOR message into a single-line message.

The following are two examples of the planning that is necessary before you code your exit routine to process a message:

- To request queuing of a message to a particular console and to eliminate queuing by routing codes, the steps are:
  1. Request queuing to a particular console
  2. Request a change to the console ID or console name
  3. Specify the desired console ID or console name
  4. Request a change to the routing codes
  5. Change the routing codes to all zeroes
- To request queuing a message by routing codes only and also change the text of the message, the steps are:
  1. Request queuing by routing codes only
  2. Change the routing code to whatever is desired
  3. Request a change in the message text
  4. Specify the new length of the text
  5. Supply the new text

Incompatible Requests: The system handles incompatible requests in one of two ways. If IEAVMXIT or an MPF exit makes conflicting requests, the message is either (1) processed in its original state or (2) processed according to the request that is least detrimental to the message.

Incompatible request errors are signaled in the "MPF request flag" field in the SYSLOG.

The following incompatible requests cause the message to be processed in its original state:

- A request to delete a message (CTXTRDTM) and a request in the request flags in CTXTRFLG that specifies processing other than a request to affect the automation (CTXTRAYS or CTXTRANO) of the message. Any requests in the extended request flags (CTXTERFS) may be specified with the request to delete a message, including the request to suppress the message from JOBLOG (CTXTESJL)
- A request to queue via routing codes only (CTXTRQRC) and a request to:
  - Queue to a particular active console (CTXTRQPC)
  - Queue to hardcopy only (CTXTRHCO)
  - Change the message type (CTXTRCMF)
- A request to queue a message to hardcopy only (CTXTRHCO) and a request to broadcast the message to active consoles (CTXTRBCA)
- A request to retain a message (CTXTRRET) and a request not to retain a message (CTXTRNRT)
- A request to automate a message (CTXTRAYS) and a request not to automate a message (CTXTRANO)
- A request to allow the primary subsystem to alter message routing (CTXTEMRY) and a request not to allow the primary subsystem to alter message routing (CTXTEMRN)

The following incompatible requests cause the message to be processed according to the request that is least detrimental to the message:

- A request to change both the name (CTXTRCNM) and the ID of the console (CTXTRCFC) to which a message is queued causes only the console name to change. If the specified console name is not valid, no change occurs, regardless of whether the specified console ID is valid.
- A request to send a message to hardcopy (CTXTRFHC) and a request to not send a message to hardcopy (CTXTRNHC) results in a hardcopy of the message.
- A request to send a message to hardcopy while allowing display at a console (CTXTRFHC) and a request to send a message only to hardcopy (CTXTRHCO) causes the message to be sent to hardcopy as well as displayed at any console to which it might be queued.
- A request to send a message to only hardcopy (CTXTRHCO) and a request not to send a message to hardcopy (CTXTRNHC) results in only sending a message to hardcopy.
- A request to broadcast a message (CTXTRBCA) and a request to not broadcast a message (CTXTRBCN) results in not broadcasting the message.

Previous Requests: In a JES3 complex, messages pass through MPF twice - once on the LOCAL processor, and once on the GLOBAL processor. If your exit is running on the GLOBAL processor, you can determine what was altered by an MPF exit on the LOCAL processor by looking at the previous request flags pointed to by CTXTPREQ.

You can observe the alterations that your exit has made to a message by examining the message in the SYSLOG. Each record in the SYSLOG is mapped by macro IHAHCLOG which is found in the SYS1.MODGEN system library. The HCLREQFL User Exit/MPF Request Flags field in each SYSLOG record indicates the actions taken against the message by MPF, an MPF exit, or by a subsystem on the Subsystem Interface (SSI). If you requested that a message be deleted, it will not be present in the SYSLOG.

If your exit made an incompatible request, this is also indicated in the HCLREQFL field.

Programming Considerations

When you code an IEAVMXIT routine or an MPF exit routine, observe the following conventions:

- Code the routine to be reentrant and serially reusable.
- Code the routine to use 31-bit addresses with AMODE 31 and allow residency above the 16 MB line with RMODE ANY.
- Do not code an installation exit that receives control for a message that the exit issues; this causes an endless loop. The exit must be coded so that when it receives control for that message, it does not issue the message again.
- Do not use services (ENQ) or perform actions (I/O) that can result in a WAIT since this might delay or even hang the message issuer and z/OS console support.
- Do not use message intensity fields when coding an MPF exit and using a multicolor screen. Message highlighting is best achieved by requesting a color change for specified message.

- If you specify message text or a message text length value that exceeds the maximum length allowed for that type of message, the system truncates the message.
- When suppressing write-to-programmer (WTP) messages, either change the routing code so that routing code 11 is not specified (CTXTRCRC is ON and CTXTR11 is OFF), or set bit CTXTNWTP ("do not do WTP processing") ON.
- Some messages are issued using the MSGTYP parameter on the WTO macro, causing the message to be routed to the consoles that requested to see the message as the result of a MONITOR command. To queue these messages by any other queueing attribute (for example, by console ID or route codes), it is necessary to zero the message type bytes. To determine if an IBM message is routed to consoles as a result of the MONITOR command, please refer to the CTXTMTYP field.
- To prevent almost all further processing of a message, set these bits ON:

  Bit **Description**

  CTXTRDTM
  "Delete the message." The message will not be displayed on consoles or logged in hardcopy.

  CTXTRANO
  "Automation is not required for this message." The message will not be sent to EMCS consoles receiving automation messages.

  CTXTESJL
  "Suppress from joblog." The message will not go into the JES job log.

  CTXTNWTP
  "Do not do WTP processing." The message will not be sent to a TSO user's terminal or to the system message data set of a batch job.

  Setting those four bits ON will prevent almost all of the usual message processing. However, the message is still shown on the message SSI. Use extreme caution when doing this to a message because there will be no record of it in the system.
- Messages can be queued exclusively to EMCS consoles that are receiving automation messages.To do this, use the CTXTRDTM, CTXTESJL, and CTXTNWTP bits. Then either use CTXTRAYS or specify AUTO(YES) in the MPFLSTxx member for a particular message.
- Do **not** add routing code 11 to message IEF170I, as this causes an endless loop.
- You must explicitly request processing for subsequent lines of a multiple-line WTO, by setting the CTXTRPML bit in the CTXT to 1.
- When processing minor lines of a multiple-line WTO, the installation exit can change only the message text of the current minor line.
- On entry, the CTXT indicates (in the CTXTSYSN field) which system sent the message.
- If the WTO/R is a branch-entry WTO/R, the CTXTNBEW bit is set to 1. This is for informational purposes only.
- IEAVMXIT and MPF exit routines will **not** be invoked during the initial processing of synchronous WTOs or WTORs. The exit routine will be invoked when the message is later issued (via SVC) to the hardcopy log.
- Some messages (such as $HASP373) have their text completed when WTO calls the subsystem interface. This call occurs after the exit routine completes its processing.

- When replying to a WTOR, you should note the following restrictions:
  - An exit routine should reply to a suppressed WTOR (using the MGCRE macro); otherwise, the WTOR remains outstanding but will not be displayed unless the operator issues a DISPLAY R command.
  - An installation exit should not request deletion of a WTOR; a request for deletion results in suppression of the WTOR. The operator will not be aware of the WTOR unless the operator issues the DISPLAY R command.
  - A WTOR may not be displayed on an MCS console when the reply is processed before the message can be displayed. A WTOR that is replied to with an exit routine can be seen in the hard-copy log.
  - IEAVMXIT or the MPF exit routine uses the following fields in the CTXT to determine the WTOR message for which it has been invoked:

    Field Description

    CTXTRPYB
    Binary representation of the message reply ID

    CTXTRPYL
    Length of the reply ID (halfword)

    CTXTRPYI
    The reply id, in EBCDIC (8 bytes, left-justified, and padded with blanks)
  - A reply issued by an MPF exit to a WTOR will appear twice on the JES job log of the job that issued the WTOR. This is because the system displays the reply once on the job log of the job that issued the WTOR and once on the job log of the job that issued the reply to that WTOR. An MPF exit replying to a WTOR runs in the address space of the job that issued the WTOR, so in this case the two jobs are the same.

Common Data Area: IEAVMXIT and all MPF exit routines receive the address of a 12-byte common data area (pointed to by CTXTCWKP in the CTXT). The common data area allows the exit routines to share data (in common work areas) across invocations.

Sharing Data With Other Exit Routines: You can code IEAVMXIT or an MPF exit routine to create work areas in the extended common storage area (ECSA), by issuing a GETMAIN or STORAGE macro, and then placing the address of these work areas in the common data area. Whenever IEAVMXIT or the MPF exit routines are invoked, the exit routines can access the common data area to obtain the work area addresses. If the exits require 12 bytes or less of data, you can place the data itself in the common data area instead of creating work areas.

The system initializes the common data area to zero; thereafter, the common data area contains whatever values the exit routines place in it.

IEAVMXIT and the MPF exit routines must manage serialization of the common data area.

Individual Data Area: In addition to the common data area, IEAVMXIT and all MPF exit routines receive the address of an 8-byte individual data area (pointed to by CTXTIWKP in the CTXT) whenever they are invoked. Each exit routine can use its individual data area to pass data (or the address of a work area) to itself across invocations.

Passing data to itself: To enable an exit routine to pass data to itself across invocations, code the routine to:

1. Create a work area in the ECSA by issuing a GETMAIN or STORAGE macro
2. Place the address of the work area in the individual data area.

During subsequent invocations, the exit routine can obtain the address of the work area by accessing its individual data area. As with the common data area, the system initializes each individual data area to zero; thereafter, the individual data area contains whatever values the exit routine places in it.

If the data required by the exit is 8 bytes or less, you can place the data itself within the individual data area instead of using a work area.

IEAVMXIT and the MPF exit routines must manage serialization of the individual data area.

Cleaning Up Work Areas: When IEAVMXIT or an MPF exit routine is to be deactivated (via CONTROL M or SET MPF), and the contents of its individual data area are nonzero, the exit routine is invoked before it is deactivated, so that it can clean up any work areas it may have created.

Individual work area should be cleaned up when the exit that owns them terminates. Common work areas should be cleaned up when the last exit using them terminates.

The exit routine determines whether it has been called for deactivation by checking the CTXTCIDA bit in the CTXT. The CTXTCIDA bit is set to 1 to indicate deactivation.

When the exit routine is reactivated, its individual data area is reset to zero by the system.

Macro Instructions and Restrictions: IEAVMXIT and MPF exit routines can issue system macros, but you should be aware of the following restrictions:

- Do not install an exit routine that issues the WAIT macro or calls a service that issues a WAIT. WAITs and implied WAITs can terminate console communications.
- Do not use macros with expansions that store information into an inline parameter list.
- Do not issue a GETMAIN or STORAGE macro for subpools that represent space within a region (0 — 127, 240, or 250 — 252). Because the exit routine executes as a part of the control program, it can use subpools such as 229, 230, and 249.

Security Consideration: It is the responsibility of your installation to provide any required security for an exit routine that issues the MGCRE macro. For example, the routine can issue the RACROUTE REQUEST=TOKENBLD macro to obtain the user token for a user ID that is authorized to the command and then append the security token to the MGCRE parameter list.

Entry Specifications

On entry, register 1 points to the address of the exit parameter list, the CTXT.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the pointer to the CTXT
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit routine

Parameter List Contents: Register 1 contains the address of a pointer to the exit parameter list (the CTXT), which is mapped by macro IEZVX100 (data area CTXT). The IEZVX100 mapping is described in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

IEAVMXIT or an MPF exit routine returns to the calling module by using a branch and return via register 14.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-15	Restored to contents at entry

Coded Examples of MPF Exit Routines

IBM provides the following examples of MPF exit routines in SYS1.SAMPLIB, which can be used to modify message processing:

- IEACWAIT — used to cancel jobs that are waiting for volumes
- IEAOCANC — used to cancel jobs that are waiting for data sets
- IEAKTRCK — used to route status messages, for critical jobs, to a particular console
- IEAJTRCK — used to track JES2 jobs that are started during a given period.

Chapter 15. IEAVTABX — Change Options / Suppress Dump Exit

Topics for This Exit Appear as Follows:

IEAVTABX is a SYSUDUMP/SYSABEND/SYSMDUMP dumping services installation exit routine name list. It contains a list of installation exit routine names to be given control before the dump actually is taken. These installation exit routines allow an installation to change the dump options in effect or to suppress the dump that would be generated by an abending task.

You can use IEAVTABX to:

- Change dump options or suppress a dump based on job name, abend code, or other information in the SDWA.
- Tailor dumps for specific problems before the dump is taken.

Installing the Exit Routine

To install a dumping services installation exit routine in your system, you need to take the following steps:

- Linkedit the dumping services exit routine into SYS1.LPALIB, SYS1.LINKLIB, or a data set in the LNKLST concatenation.
- Define the dumping services exit to MVS by adding its load module name to the exit name list in IEAVTABX.

Defining Dumping Services Exits to IEAVTABX

You can specify dumping services exits in any of the blank entries in the IEAVTABX exit name list. At the completion of each SVC DUMP or SYSMDUMP, the system invokes the routines in the order you specified them. That is, the exit routine specified in entry 1 gets control first, followed by the exit routine specified in entry 2, and so on.

IEAVTABX is a CSECT in load module IEAVTABX (which resides in SYS1.LPALIB). The IBM-supplied version of IEAVTABX contains the following entries:

- A 4-byte count field that contains hexadecimal zeroes, followed by
- Ten 8-byte entries that contain EBCDIC blanks, and are intended for customer use, followed by
- An 8-byte entry that contains eight character zeroes (X'F0') to indicate the end of the exit name list (the end-of-table marker).

You can add as many dumping services exit routine load modules to IEAVTABX as you want. IBM recommends that you use the existing blank entries in IEAVTABX before creating new ones. Ensure that the last entry is an 8-byte field that contains eight character zeroes (X'F0').

Contents of Entries: The first 4-byte entry contains the count field, which indicates the number of exit routine names in the table. Each 8-byte entry after the count field contains the exit load module name.

This is how the first, second, third and end-of-table marker fields in the IEAVTABX supplied by IBM appear:

```
IEAVTABX
HEX LOCATION:     CONTENTS:
0000              00000000              COUNT
0004              40404040 40404040     ENTRY 1
000C              40404040 40404040     ENTRY 2
 .                          .            .
 .                          .            .
 .                          .            .
0054              F0F0F0F0 F0F0F0F0     END OF LIST
```

To insert a dumping services exit load module name in the list, select an available entry (one that is set to blanks) and substitute the exit load module name in place of the blanks. Exit names can be one to eight characters.

Examples of changing entries in IEAVTABX are shown in the sections that follow .

Adding and Deleting Exit Names in IEAVTABX: To add or delete exit names in the exit name list, you must modify the IEAVTABX object code in SYS1.LPALIB through the use of the SPZAP program, or through SMP/E. For more information, refer to "Using SPZAP or SMP/E to Add or Delete Name."

Using SPZAP or SMP/E to Add or Delete Name

Adding Exit Names: When using SPZAP or SMP/E, take the following steps:

- Use SPZAP to produce a dump of IEAVTABX. Sample JCL follows:

```
//DUMPJCL  JOB MSGLEVEL=(1,1)
//STEP     EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSNAME=SYS1.LPALIB,DISP=OLD
//SYSIN    DD *
 NAME IEAVTABX
 DUMP IEAVTABX
/*
```

- Use the dump produced to select an available entry in IEAVTABX.
- Write the EBCDIC name of the dumping services exit module in the entry. (See Example 1 if you are using SPZAP or Example 1A if you are using SMP/E.)

The changes will take affect on the next IPL.

Example 1 - Using SPZAP to Add Exit Names: The following job adds EXITRTN1 to the first entry in IEAVTABX, and EXITRTN2 to the second entry in IEAVTABX. The job then dumps IEAVTABX to verify the changes.

```
//EXAMPLE1 JOB MSGLEVEL=(1,1)
//STEP     EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSNAME=SYS1.LPALIB,DISP=OLD
//SYSIN    DD *
 NAME IEAVTABX
 VER 0000 0000,0000              COUNT FIELD
 REP 0000 0000,0002              SET COUNT FIELD
 VER 0004 4040,4040,4040,4040    FIRST UNUSED ENTRY
 REP 0004 C5E7,C9E3,D9E3,D5F1    SET TO EXITRTN1
 VER 000C 4040,4040,4040,4040    SECOND UNUSED ENTRY
 REP 000C C5E7,C9E3,D9E3,D5F2    SET TO EXITRTN2
 DUMP IEAVTABX
/*
```

Example 1A - Using SMP/E to Add Exit Names: The following job adds EXITRTN1 to the first entry in IEAVTABX, and EXITRTN2 to the second entry in IEAVTABX.

```
++USERMOD(USRMOD2).
++VER(Z038) FMID(HBB4410). /* CHANGE THE FMID AS NEEDED */
++ZAP(IEAVTABX).
 NAME IEAVTABX IEAVTABX
 VER 0000 0000,0000              COUNT FIELD
 REP 0000 0000,0002              SET COUNT FIELD
 VER 0004 4040,4040,4040,4040    FIRST UNUSED ENTRY
 REP 0004 C5E7,C9E3,D9E3,D5F1    SET TO EXITRTN1
 VER 000C 4040,4040,4040,4040    SECOND UNUSED ENTRY
 REP 000C C5E7,C9E3,D9E3,D5F2    SET TO EXITRTN2
```

Deleting Exit Names: When using SPZAP or SMP/E, take the following steps:

- Use SPZAP to dump IEAVTABX (as shown earlier in the sample JCL code under "Using SPZAP or SMP/E to Add or Delete Name" on page 94).
- Write EBCDIC blanks in place of the dump processing exit module name. (See Example 2 if you are using SPZAP or Example 2A if you are using SMP/E.)

The changes will take effect on the next IPL.

If all entries are in use, you can replace module IEAVTABX, or expand it. Ensure that the last entry contains 8 bytes of hexadecimal zeroes.

Example 2 - Using SPZAP to Delete Exit Names: The following job deletes EXITRTN1 from the dump processing exit list. The job then dumps IEAVTABX to verify the changes.

```
//EXAMPLE2 JOB MSGLEVEL=(1,1)
//STEP     EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSNAME=SYS1.LPALIB,DISP=OLD
//SYSIN    DD *
 NAME IEAVTABX
 VER 0000 0000,0002              COUNT FIELD
 REP 0000 0000,0001              SET COUNT FIELD
 VER 0004 C5E7,C9E3,D9E3,D5F1    EXITRTN1
 REP 0004 4040,4040,4040,4040    RESTORE ENTRY 1
 DUMP IEAVTABX
/*
```

Example 2A - Using SMP/E to Delete Exit Names: The following job deletes EXITRTN1 from the dump processing exit list.

```
++USERMOD(USRMOD3).
++VER(Z038) FMID(HBB4410). /* CHANGE THE FMID AS NEEDED */
++ZAP(IEAVTABX).
 NAME IEAVTABX
 VER 0000 0000,0002              COUNT FIELD
 REP 0000 0000,0001              SET COUNT FIELD
 VER 0004 C5E7,C9E3,D9E3,D5F1    EXITRTN1
 REP 0004 4040,4040,4040,4040    REPLACE WITH BLANKS
```

See the following references for more information on the use of:

- SPZAP, see *z/OS MVS Diagnosis: Tools and Service Aids*.
- SMP, see *SMP/E for z/OS User's Guide*.
- Expand, see *z/OS MVS Program Management: User's Guide and Reference*.

Exit Routine Environment

Each routine in IEAVTABX must reside in (E)PLPA, and receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31 and RMODE ANY.
- With no locks held.
- Under the abending task and in the home address space.
- Under the SVRB of ABDUMP.

Exit Recovery: Each exit routine must establish an ESTAE and request a tailored dump. Before each exit routine returns control to ABDUMP it must delete the ESTAE, and it must free all storage it obtained.

Note: A request for an ABDUMP from the ESTAE will cause recursion, and no dump will be produced for the installation exit routine error. It is suggested that you take an SDUMP and issue a SETRP DUMP=NO. These actions cause a retry attempt to a return point that prevents ABDUMP's recovery routine from getting control.

Exit Routine Processing

The installation exit routines receive control sequentially prior to taking the dump. At entry, each routine has addressability to a parameter list that is the major communication area among the installation exit routines. The input parameter list is mapped by the macro IHAABEPL (data area ABEP) and contains a copy of the following types of information for each dump:

- Job name
- System completion code
- Address of the SDWA
- Module name
- Options in effect (SNAP parameter list)
- Parameter list level indicator
- Return code from the previous installation exit routine

The SNAP parameter list, mapped by IHASNAPX (data area SNAPX), contains the current dump options in effect. You can change these options in your exit routine based on other information in the ABEPL and the SDWA.

The exit routine get control prior to taking the dump. If one of the exit routines suppresses the dump, the system issues message IEA848I, indicating that dump suppression has taken place.

Programming Considerations

The installation exit routines in IEAVTABX must be reentrant. Their load module names must be eight bytes padded to the right with blanks.

The exits must obtain storage from specific subpools requested in the tailored dump. Before returning, each exit routine must free all the storage it obtained.

The installation exit routine must not free the entry parameter list.

Entry Specifications

The ABDUMP calling routine passes to each installation exit routine in IEAVTABX the address of a parameter list used for communication between the routines.

Registers at Entry: The contents of the registers on entry to an installation exit routine specified in IEAVTABX are:

Register	Contents
0	Not applicable
1	Address of IHAABEPL parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the installation exit routine in IEAVTABX

Parameter List Contents: Register 1 points to the exit routine parameter list, which is mapped by the IHAABEPL macro (data area ABEP). For a mapping of the ABEP data area, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

Each installation exit routine returns a code indicating whether processing should continue.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents at entry
15	One of the following return codes:

Return Code	Explanation
0	Continue processing with the current options.
4	Change options as indicated in IHAABEPL.
8	Suppress the dump.

Chapter 16. IEAVTSEL — Post Dump Exit Name List Exit

Topics for This Exit Appear as Follows:

IEAVTSEL is the SVC, IEATDUMP, and SYSMDUMP post-dump exit names list. In IEAVTSEL, you list the names of installation exit routines that the system is to invoke after each SVC dump and SYSMDUMP. You can also set a flag bit for each exit routine to indicate whether the system is to invoke the routine for any dump that dump analysis and elimination (DAE) has suppressed.

Each routine receives control in the order it is listed in IEAVTSEL. However, to prevent endless recursion processing loops, the system does not invoke any of the exit routines in IEAVTSEL if an SVC dump request is issued by a post-dump exit routine or its recovery routine.

You can use post-dump exit routines to:

- Extract certain information from the header record, such as the dump title, ERRORID, time of dump, ABEND code, and failing module name. This information can be written to a log data set (using DISP=MOD to append the new entries at the end).
- Offload the dump from SYS1.DUMPnn to a DASD or tape data set for later processing and log the information in a log data set.
- Start an IPCS job that would print a small portion of the dump (such as log data, summary, and/or SUMDUMP).

Installing Post-Dump Exit Routines

To install a post-dump installation exit routine in your system, you need to take both of the following steps:

- Linkedit the post-dump exit routine into SYS1.LPALIB, SYS1.LINKLIB, or a data set in the LNKLST concatenation. The library containing the exit routine must be APF-authorized.
- Define the post-dump exit to MVS by adding its load module name to the exit name list in IEAVTSEL.

Defining Post-Dump Exits to IEAVTSEL

You can specify post-dump exits in any of the blank entries in the IEAVTSEL exit name list. At the completion of each SVC dump, IEATDUMP, or SYSMDUMP, the system invokes the routines in the order you specified them. That is, the exit routine specified in entry 1 gets control first, followed by the exit routine specified in entry 2, and so on.

IEAVTSEL is a CSECT in load module IEAVTSEL (which resides in SYS1.LINKLIB). The IBM-supplied version of IEAVTSEL contains the following 12-byte entries:

- Nine entries that contain EBCDIC blanks, and are intended for customer use, followed by
- A number of entries that are reserved for use by IBM, followed by
- A final entry that contains hexadecimal zeroes to indicate the end of the exit name list (the end-of-table marker).

You can add as many post-dump exit routine load modules to IEAVTSEL as you want. IBM recommends that you use the existing blank entries in IEAVTSEL before creating new ones. Ensure that you do not delete any entries in use by MVS, and that the last entry contains 12 bytes of hexadecimal zeroes.

Contents of Entries: The first 8 bytes of each entry contain the exit load module name, and the last 4 bytes of each entry contain a flag bit you would set under certain conditions (explained in "The Flag Bit"). Entries that have 8 bytes of X'40' (blanks) for the exit load module name are available for customer use.

This is how the first, second, and last entries in IEAVTSEL appear:

```
IEAVTSEL
HEX LOCATION:     CONTENTS:
0000              40404040 40404040 00000000      ENTRY 1
000C              40404040 40404040 00000000      ENTRY 2
 .                          .                        .
 .                          .                        .
 .                          .                        .
0078              00000000 00000000 00000000      END OF LIST
```

To insert a post-dump exit load module name in the list, select an available entry (one that is set to blanks) and substitute the exit load module name in place of the blanks. Exit names can be one to eight characters. Also, set the flag bit if appropriate.

For examples of changing entries in IEAVTSEL, see "Methods of Adding and Deleting Exit Names in IEAVTSEL" on page 101.

The Flag Bit

You can set the high-order bit of the 4-byte flag field to indicate whether the corresponding exit routine is invoked for dumps that are suppressed through dump analysis and elimination (DAE).

- If the bit is off (X'00000000'), the exit routine gets control when dump processing ends, if DAE has ***not*** suppressed the dump.
- If the bit is on (X'80000000'), the exit routine gets control when dump processing ends, even if DAE has suppressed the dump.

Methods of Adding and Deleting Exit Names in IEAVTSEL

There are two ways to add or delete exit names in the exit name list. You can use either of the following methods:

- **Method 1:** Modify the IEAVTSEL source code through the use of the DUMPEXIT macro. Note that DUMPEXIT is defined within the IEAVTSEL module. To use DUMPEXIT, you need the source code for IEAVTSEL. You can obtain the source code for IEAVTSEL from the optional machine-readable material provided by IBM. For additional information on this optional material, contact your IBM Marketing representative.
- **Method 2:** Modify the IEAVTSEL object code in SYS1.LINKLIB through the use of the SPZAP program, or through SMP/E.

Both methods are explained in greater detail in the sections that follow .

Method 1 — Using DUMPEXIT to Add Names: When using the DUMPEXIT macro, take the following steps:

- Select an available entry in IEAVTSEL, which appears in IEAVTSEL as:

  ```
  DUMPEXIT NAME=,ATTR=00000000
  ```

- On the NAME parameter of DUMPEXIT, specify the load module name of the exit routine to be added and set the flag field if desired (on the ATTR parameter). In this example:

  ```
  DUMPEXIT     NAME=EXITRTN,ATTR=xxxxxxxx
  ```

  'EXITRTN' is the load module name of the post-dump exit routine to be added, and 'xxxxxxxx' is either X'00000000' or X'80000000', depending on whether you want the exit routine to receive control for dumps that are suppressed through DAE.

  Ensure that you do not delete any entries in use by MVS, and that the last entry contains 12 bytes of hexadecimal zeroes.

- Assemble and linkedit the IEAVTSEL exit again.

The changes will take effect on the next IPL.

You can add as many post-dump exit routine load modules to IEAVTSEL as you want. Additional DUMPEXIT macro invocations can be added to the post-dump exit routine load module anywhere ***before*** the end of the table marker:

```
DC    XL12X'000000000000000000000000'
```

Method 1 — Using DUMPEXIT to Delete Names: When using the DUMPEXIT macro, take the following steps:

- Replace the entry to be deleted with:

  ```
  DUMPEXIT NAME=,ATTR=00000000
  ```

- Assemble and linkedit IEAVTSEL again.

The changes will take effect on the next IPL.

Method 2 — Using SPZAP or SMP/E to Add Names: When using SPZAP or SMP/E, take the following steps:

- Use SPZAP to produce a dump of IEAVTSEL. The sample JCL code follows:

  ```
  //DUMPJCL  JOB MSGLEVEL=(1,1)
  //STEP     EXEC PGM=AMASPZAP
  //SYSPRINT DD SYSOUT=*
  ```

```
//SYSLIB   DD DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSIN    DD *
 NAME IEAVTSEL
 DUMP IEAVTSEL
/*
```

- Use the dump produced to select an available entry in IEAVTSEL.
- Write the EBCDIC name of the post-dump exit module in the entry and set the appropriate flag setting. (See Example 1 if you are using SPZAP or Example 1A if you are using SMP/E.)

The changes will take affect on the next IPL.

Example 1 - Using SPZAP to Add Exit Names: The following job adds EXITRTN1 to the first entry in IEAVTSEL and sets the flag to indicate that EXITRTN1 should get control even if the dump is suppressed by DAE. The job also adds EXITRTN2 to IEAVTSEL and sets the flag to indicate that EXITRTN2 should get control only when a dump is not suppressed by DAE. The job then dumps IEAVTSEL to verify the changes.

```
//EXAMPLE1 JOB MSGLEVEL=(1,1)
//STEP     EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSIN    DD *
 NAME IEAVTSEL
 VER 0000 4040,4040,4040,4040,0000,0000    FIRST UNUSED ENTRY
 REP 0000 C5E7,C9E3,D9E3,D5F1,8000,0000    SET TO EXITRTN1
 VER 000C 4040,4040,4040,4040,0000,0000    SECOND UNUSED ENTRY
 REP 000C C5E7,C9E3,D9E3,D5F2,0000,0000    SET TO EXITRTN2
 DUMP IEAVTSEL
/*
```

Example 1A - Using SMP/E to Add Exit Names: The following job adds EXITRTN1 to the first entry in IEAVTSEL and sets the flag to indicate that EXITRTN1 should get control even if the dump is suppressed by DAE. The job also adds EXITRTN2 to IEAVTSEL and sets the flag to indicate that EXITRTN2 should get control only when a dump is not suppressed by DAE.

```
++USERMOD(USRMOD2).
++VER(Z038) FMID(HBB4410). /* CHANGE THE FMID AS NEEDED */
++ZAP(IEAVTSEL).
 NAME IEAVTSEL IEAVTSEL
 VER 0000 4040,4040,4040,4040,0000,0000    FIRST UNUSED ENTRY
 REP 0000 C5E7,C9E3,D9E3,D5F1,8000,0000    SET TO EXITRTN1
 VER 000C 4040,4040,4040,4040,0000,0000    SECOND UNUSED ENTRY
 REP 000C C5E7,C9E3,D9E3,D5F2,0000,0000    SET TO EXITRTN2
```

Method 2 — Using SPZAP or SMP/E to Delete Name: When using SPZAP or SMP/E, take the following steps:

- Use SPZAP to dump IEAVTSEL. Sample JCL code for using SPZAP or SMP/E to add or delete names is shown in “Methods of Adding and Deleting Exit Names in IEAVTSEL” on page 101.
- Write EBCDIC blanks in place of the post-dump exit module name and set the flag bit off. (See Example 2 if you are using SPZAP or Example 2A if you are using SMP/E.)

The changes will take effect on the next IPL.

If all entries are in use, you can replace module IEAVTSEL, or expand it. Ensure that you do not delete any entries in use by MVS, and that the last entry contains 12 bytes of hexadecimal zeroes.

Example 2 - Using SPZAP to Delete Exit Names: The following job deletes EXITRTN1 from the post-dump exit list. The job then dumps IEAVTSEL to verify the changes.

```
//EXAMPLE2 JOB MSGLEVEL=(1,1)
//STEP     EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSIN    DD *
 NAME IEAVTSEL
 VER 0000 C5E7,C9E3,D9E3,D5F1,8000,0000    EXITRTN1
 REP 0000 4040,4040,4040,4040,0000,0000    RESTORE ENTRY 1
 DUMP IEAVTSEL
/*
```

Example 2A - Using SMP/E to Delete Exit Names: The following job deletes EXITRTN1 from the post-dump exit list.

```
++USERMOD(USRMOD3).
++VER(Z038) FMID(HBB4410). /* CHANGE THE FMID AS NEEDED */
++ZAP(IEAVTSEL).
 NAME IEAVTSEL IEAVTSEL
 VER 0000 C5E7,C9E3,D9E3,D5F1,8000,0000    EXITRTN1
 REP 0000 4040,4040,4040,4040,0000,0000    REPLACE WITH BLANKS
```

For more information on the following topics, use these references:

- SPZAP, see *z/OS MVS Diagnosis: Tools and Service Aids*.
- SMP, see *SMP/E for z/OS User's Guide*.
- Expand, see *z/OS MVS Program Management: User's Guide and Reference*.

Exit Routine Environment

Each routine in IEAVTSEL receives control in the following environment:

- Enabled for interrupts.
- In supervisor state, PSW key 0.
- In AMODE 24 or 31 and RMODE ANY.
- In task mode in the DUMPSRV (dumping services) address space.
- With no locks held.

Exit Recovery: SVC dump establishes its own ESTAE before calling the IEAVTSEL exit routines. Each exit routine must establish its own recovery.

Exit Routine Processing

The installation exit routines listed in IEAVTSEL receive control with the completion of the dump.

Each exit routine in IEAVTSEL accesses a common parameter list (SDEPL) containing data and an interface area that the routines can use to pass information to succeeding routines. The SDEPL is mapped by macro IHASDEPL.

Exit Status Flags: A post-dump exit routine listed in IEAVTSEL can determine whether the exit routine that immediately preceded it completed successfully or not by checking the SDEPLEXE flag bit in the SDEPL. The system sets this bit on if the preceding exit routine returned a nonzero return code (in register 15).

A post-dump exit routine can determine whether **any** of the preceding exit routines failed to complete successfully by checking the SDEPLERR flag bit in the SDEPL. Like SDEPLEXE, this bit is set to '1' if a post-dump exit returns a nonzero return code. However, once set on, SDEPLERR is not reset when a subsequent exit routine returns a zero return code.

Programming Considerations

Parameter List: The SDUMP exit parameter list (SDEPL) is passed to each post-dump exit routine listed in IEAVTSEL. In addition to other useful information, the SDEPL contains the addresses of both an exit interface area and a 200-byte exit work area:

- The exit interface area (pointed to by SDEPLEXT) enables a post-dump exit routine to pass information to successive post-dump exit routines. This area is set to zeroes before the call to the first post-dump exit routine. Thereafter, the exit interface area contains whatever values the post-dump exit routines place in it. If an installation chooses, it can use this area as a work area
- The exit work area (pointed to by SDEPLWA) is a general work area for the installation exit routine. This area is reset to zeroes between calls to each exit named in IEAVTSEL

Dump Header Record: The SDEPLHD field in the parameter list (SDEPL) points to a copy of the dump header record. The AMDDATA mapping macro maps the dump header record, and the description of AMDDATA contains information about fields in the header record. For details, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

The DAE section of the dump header record, which is located in the second 2K of the header record, contains information about the dump that DAE has gathered. Some fields in the DAE section of the header record that contain useful diagnostic information are:

ADSSDAE
Start of the DAE section of the header record.

DAESSMVS
Symptom string used for matching.

DAECRIT
Criteria for unique symptom string generated by DAE.

DAESTAT
DAE status flags mapped by ADYDSTAT.

DAEERID
ERROR-ID from the original occurrence of the dump.

DAEDCNT
The number of occurrences of the dump.

If DAE has not suppressed the dump, the first 2K of the dump header record also contains information that a post-dump exit routine can use. To determine whether DAE has suppressed the dump, the exit routine can test the PRDID field in the

header record. If the PRDID field contains X'000000', DAE has suppressed the dump, and the first 2K of the header record does not contain useful information. If PRDID is non-zero, DAE has not suppressed the dump, and the first 2K of the dump header record contains valid information about the dump. Some fields in this part of the header record that contain useful diagnostic information are:

PRDDUMPT
PRDDUMPT Dump type that tells whether the dump is a stand alone dump, an SVC dump, a SYSMDUMP, or an SVC dump for a SLIP request.

PRDDSNAM
Dump data set name to which dump was taken.

PRDERRID
Error ID from this dump.

PRDSDWA
A copy of the SDWA of the caller of SDUMP. From this SDWA you can obtain the failing module name, the ABEND code, and any other diagnostic data. See mapping macro SDWA in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for available fields.

The SDWA is not present on SLIP dumps.

Other Considerations:

- Dump data sets are allocated with the following attributes:
  - RECFM = FBS
  - LRECL = 4160
  - BLKSIZE = whatever is optimum size per track

  Therefore, have your exit routines specify these attributes when defining DCBs to use to open SYS1.DUMPnn data sets.
- If an installation does not want to have the installation exit routine run in supervisor state or key 0, the exit routine must issue a MODESET macro to obtain the desired state.

Entry Specifications

Each exit routine in IEAVTSEL receives control sequentially and receives a common parameter list.

Registers at Entry: The contents of the registers on entry to an installation exit routine in IEAVTSEL are as follows.

Register	Contents
0	Not applicable
1	Address of the address of the parameter list mapped by IHASDEPL
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the installation exit routine in IEAVTSEL

Parameter List Contents: Register 1 points to address of the SDUMP exit parameter list (SDEPL) mapped by macro IHASDEPL (data area SDEPL). For a mapping of the SDEPL data area, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

The installation exit routine returns a code indicating whether the exit was successful.

Registers at Exit: Upon return from an installation exit routine in IEAVTSEL, the register contents must be as follows.

Register
Contents

0-14 Restored to contents at entry

15 One of the following return codes

Return Code
Explanation

0 The exit was successful.

nonzero
The exit was unsuccessful.

The system sets exit status flags SDEPLEXE and SDEPERR based on the return code in register 15. See “Exit Routine Processing” on page 103.

Chapter 17. IEF_ALLC_OFFLN — Allocated or Offline Device Installation Exit

Topics for This Exit Appear as Follows:

When a job must wait because a device it requested is offline or allocated to another job, MVS issues WTORs that instruct the system operator to take one of the following actions:

- Cancel the waiting job
- Bring the device online
- Allow the job to wait for the device to become available.

You can automate your installation's responses to allocation requests for offline, pending offline, or allocated devices, and reduce the need for operator intervention by:

- Defining an installation default policy for handling the ***majority*** of potential allocation requests for offline, pending offline, or allocated devices. Specify the default policy in the ALLOCxx member of SYS1.PARMLIB.
- Coding the Allocated or Offline Device exit routine to make exceptions, if any, to the installation default policy for certain jobs and/or devices. You can specify the exit in the EXITxx or PROGxx member of SYS1.PARMLIB; however, IBM recommends that you use PROGxx.

For a list of the allocation messages you can automate or suppress, see "Message Processing" on page 114.

Using the information it receives about the job and the required device(s), the exit routine can:

- Cancel the job
- Cause offline devices to be brought online to satisfy the job's request
- Cause a pending offline device to be considered for allocation
- Allow the job to wait while:
  - holding resources
  - not holding resources
- Allow the installation's default policy to determine which action to take.
- Allow the WTOR to be issued so that the system operator can decide how to handle the job. If the WTOR is to be issued, the exit routine can determine which device numbers will be displayed on the WTOR.

  Note: The exit routine can only exclude device numbers from the WTOR. It cannot exclude the actual list of devices eligible for allocation.

For more information about the ALLOCxx, EXITxx, and PROGxx parmlib members, see *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the Allocated or Offline installation exit to the dynamic exits facility. You can refer to the exit by the name IEF_ALLC_OFFLN. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Replacing the Exit Routine

For information about replacing a dynamic exit routine, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

The exit routine receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1.
- In AMODE 31 and RMODE ANY.
- With no locks held. (However, it may hold an exclusive ENQ on major name SYSZTIOT for the address space in which the allocation occurs.)

Exit Recovery

The exit routine should provide its own recovery. If the exit routine abnormally terminates, its recovery routine will get control.

If the exit routine abnormally terminates, and the exit routine does not provide its own recovery, or the error percolates beyond the exit's recovery routine, a system recovery routine will get control and fail the allocation request.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

MVS invokes the allocated or offline device exit routine or routines, if any are specified to the dynamic exits facility, every time a job must wait for a device because all devices that could satisfy the job's request are either offline, pending offline, or allocated to other jobs. This exit is only invoked for tape requests and non-SMS-managed DASD requests. It is not invoked for SMS-managed DASD requests.

MVS invokes the exit routine before issuing WTORs that:

- Identify the job that is waiting
- List the devices that are unavailable
- Request operator action.

These messages are listed in "Message Processing" on page 114.

Using the Information in the Parameter List: MVS passes the address of a list of parameters to the exit routine. The parameters contain the following information:

- Job name
- Step name
- Name of the DD statement that requires the resource
- Name of the data set that requires the resource
- If specific volumes are needed, the serial numbers of the volumes
- The number of nonspecific scratch volumes
- The number of nonspecific private volumes
- A list of the eligible devices that can be brought online

  Note: The list may include pending offline devices. You can use this exit to indicate whether these devices are eligible for allocation. If a pending offline device is permanently resident DASD or has a reserved volume mounted, its entry in the list contains the mounted volume serial number.
- An indication of whether the job can wait
- An indication of whether the job can bring devices online
- An indication of whether this is a repeated call to the exit
- Action to be taken in response to WTOR (this field, ACTION, is filled in by the exit routine).
- The number of 'wait without holding resources' decisions that the system will allow to be made for a particular device request (in the WAITNOHC field). This number includes decisions made by both the exit routine and the installation default. This number does not restrict the number of 'wait without holding resources' decisions that can be made by the system operator.
- An indication of whether the system-managed tape library is online.
- An indication of whether the system-managed tape library is offline.
- An indication of whether the system-managed tape library is pending offline.

- For a system-managed tape library request, the name of the library.
- The relative concatenation number of the DD.
- An indication of the device class of the DD; one of the following:
  - Tape device
  - Communications device
  - Direct access device
  - Graphics display device
  - Unit record device
  - Character reader device

  If the request is for an esoteric group name that includes both tape and direct access devices, both the tape and direct access indicators will be set.

Using the information in the parameter list, the exit routine determines how the system should respond to the allocation request. The exit routine indicates its decision to the system by placing a value in the ACTION field of the parameter list.

See "Return Specifications" on page 115 for the specific values the exit routine can return.

Bringing a Device Online

The system indicates to the exit routine that the job can bring devices online by setting the OKONLINE bit to 1 (in the exit parameter list).

If the exit routine attempts to bring a device online when the OKONLINE bit is not set to 1, the system will ignore the exit routine's decision and use the installation default policy (specified in the ALLOCxx parmlib member) to determine how to respond to the allocation request. If your installation does not define a default policy for handling allocated or offline device requests, or if no ALLOCxx parmlib member is defined, the system will issue WTORs so that the system operator must respond to the job's allocation request.

The exit routine brings device(s) online by:

- Setting the ACTION field to X'08'.
- Selecting the device(s) from the offline device table (as described in "The Offline Device Table" on page 111).

Selecting an Offline Device to Bring Online: If you plan to use the exit routine to cause devices to be brought online, code the routine to check the input bits in the UXSTATUS field of the offline device table. Before selecting the device(s), check:

- The UXOFFLNE bit indicates whether the device has been varied offline (for maintenance, for example). If a device has been varied offline, the system sets the UXOFFLNE bit to 1.
- The UXNOTACC bit indicates whether the device is accessible to the exit routine. If the device is not accessible (not physically defined in the system), the system sets the UXNOTACC bit to 1.

  The exit routine can cause only accessible devices to be brought online.
- The UXPENDNG bit indicates whether the device is pending offline. If a device is pending offline it cannot be brought online by the exit. However, it can be selected for allocation consideration. See "Selecting a Pending-Offline Device for Allocation Consideration" on page 111.

- The UXVCOFFL bit indicates whether the device was varied offline by a configuration manager (for example, ESCON® Manager). If the exit routine attempts to bring the device online, the device will be brought online.
- The UXVLOFFL bit indicates whether the device is offline because it resides in an offline system-managed tape library. If the system-managed tape library is offline, the system sets the UXVLOFFL bit to 1.

The exit routine brings the device online by setting the UXONLINE bit in the UXSTATUS field to 1.

Be aware that if the chosen devices cannot be successfully brought online, Allocation will retry the allocation request anyway. If the allocation request still cannot be satisfied, the exit routine might be driven again. The parameter list does not indicate that the device was not able to be brought online previously. The exit routine should consider this when choosing devices, and should be aware that it could cause a loop if it repeatedly chooses devices that cannot be brought online.

Selecting an Eligible System-Managed Tape Library Device

The system indicates whether a request is a system-managed tape library request by setting the LBREQIND bit to one in the exit parameter list. For system-managed tape library requests, the exit or installation default policy determines whether:

- The exit is to select an eligible system-managed tape library device. The exit varies the device online only when the named library is already online.
- The operator is to select an eligible system-managed tape library device. To vary the device online, the operator must first ensure that the named library is online by issuing the DISPLAY SMS,LIBRARY command (described in *z/OS MVS System Commands*). If the library is offline or pending offline, the operator must vary the library online before varying the device online by issuing the VARY SMS,LIBRARY command (described in *z/OS MVS System Commands*).

An eligible device in a system-managed tape library might be offline for one of the following reasons:

- The device resides in a system-managed tape library that is offline or pending offline
- There are no paths to the device
- A reason other than a VARY LIBRARY offline command; for example, because the operator varied the tape device offline

Selecting a Pending-Offline Device for Allocation Consideration

The exit can be used to indicate whether a specific pending-offline device will be considered for allocation. A pending-offline device will be allocated only if no other online device becomes available. The device will remain in pending offline status.

The Offline Device Table

The offline device table (pointed to in the exit parameter list) contains the device numbers of all offline or pending offline devices that match the device type that the job specified in the allocation request.

The system sets the first 4 bytes of the offline device table to the number of entries (devices) in the table. The 4-byte field is followed by one 12-byte entry for each offline device:

Bytes	Contents
1-4	contain the device number (in EBCDIC).
5	the UXSTATUS field contains information about the offline status of the device.
6	reserved.
7-12	contain the VOLSER of the pending offline permanently resident DASD or reserved device is a volume is mounted.

The format of the offline device table follows:

Number of Entries	(4 bytes)			
Device Number	(4 bytes)	Status (1 byte)	Reserved (1 byte)	Pending-offline Volser (6 bytes)
Device Number	(4 bytes)	Status (1 byte)	Reserved (1 byte)	Pending-offline Volser (6 bytes)
⋮		⋮		
Device Number	(4 bytes)	Status (1 byte)	Reserved (1 byte)	Pending-offline Volser (6 bytes)

Letting the Job Wait for the Device

The system indicates to the exit routine (in the OKTOWAIT bit of the exit parameter list) whether the job is allowed to wait. If OKTOWAIT is set to 1, the exit routine can cause the job to wait by setting the ACTION field to indicate one of the following:

- The job will wait holding any resources it may have obtained
- The job will wait without holding any resources it may have obtained.

If the exit routine attempts to cause the job to wait when OKTOWAIT is set to 0, the system will ignore the exit routine's decision and use the installation default policy (specified in the ALLOCxx parmlib member) to determine how to respond to the allocation request. If your installation does not define a default policy for handling allocated or offline device requests, or if no ALLOCxx parmlib member is defined, the system will issue messages (listed in “Message Processing” on page 114) so that the system operator must respond to the job's allocation request.

If the exit routine allows the job to wait, the system will issue an eventual action message (IEF289E) to inform the operator that the job is waiting for a device.

Using the Exit with Your Installation's Default Policy

If you code the exit, use it in conjunction with your installation default policy for jobs that must wait for allocated or offline devices. Define your installation's default policy by specifying one of the following parameters on the POLICY keyword of the ALLC_OFFLN statement:

Parameter
Action

WTOR
Allow the messages to be issued so that the system operator must decide whether to cancel the job or let the job wait.

Note: In a sysplex environment, you will want to reduce the number of WTORs; this exit might be a candidate for that consideration.

WAITHOLD
Allow jobs to wait for devices while holding obtained resources.

WAITNOH
Allow jobs to wait for devices without holding obtained resources.

CANCEL
Cancel jobs that must wait for allocated or offline devices.

When you have chosen a default policy to handle the majority of possible requests for allocation of pending offline, offline, or already-allocated devices use the exit routine to make exceptions, if any, for certain jobs and/or devices. The exit routine's decisions will override the installation's default policy.

If you do not code the exit routine, MVS will use your installation's default policy (specified in the ALLOCxx parmlib member) to determine how to respond to ***all*** allocation requests for allocated pending offline, or offline devices. If your installation does not define a default policy, the system will always issue the WTORs.

Programming Considerations

Observe the following conventions when coding the Allocated or Offline Device Exit routine:

- Code the exit routine to be reentrant.
- Do not code the exit routine to issue dynamic allocation calls.
- Do not code an allocated or offline device exit routine if the decision of the exit routine will always be the same regardless of which devices are needed. Instead, allow your installation's default policy to determine how to handle the allocation request.
- The exit is called every time a job requires a device that is either offline, pending offline or allocated to another job. Therefore, when coding the exit routine, you should be aware that an increased path length will increase processor utilization and may degrade performance.
- When the exit routine determines that the system should issue WTORs (by setting the ACTION field to X'40'), the routine can modify the list of device numbers that will be displayed, via WTOR, to the system operator. The exit routine can exclude certain device numbers from the WTOR by setting the UXEXCLUD bit of the UXSTATUS field for the device (in the offline device table) to 1.
- When using system-managed tape libraries, it is possible to loop between offline recovery and the exit. In this situation, offline recovery calls the exit, which selects a tape device that is offline because it is in an offline system-managed tape library. However, offline recovery cannot bring the selected device online until the operator brings the library online. Therefore, the device is not removed from the table of offline eligible devices, and offline recovery again calls the exit. This looping will occur up to the number of times specified by the MAXNWAIT parameter of the ALLOCxx parmlib member. Then the system will use the installation-defined default action.

- Before using the UXOFLPTR pointer field in the input parameters, check the field's value to see if there are offline devices eligible to be allocated. A UXOFLPTR value of zero indicates that no offline devices are eligible for this request, and only already allocated devices are eligible.
- Use the exit routine, in conjunction with the installation default policy, to automate your installation's responses to WTORs such as the following:
  - IEF157E
  - IEF238D
  - IEF244I
  - IEF433D
  - IEF434D
  - IEF490I.
- When the exit routine requests to bring one or more devices online by setting the ACTION field to X'08' the system will attempt to bring the requested device(s) online, and then retry the allocation. The system will retry the allocation regardless of whether the devices can be brought online, and this might result in the exit routine being called again for the same allocation request. If the exit routine chooses the same devices to bring online, a loop will result where Allocation repeatedly calls the exit routine to attempt to satisfy the request, and the exit routine takes an action that will be unsuccessful and so the request can never be satisfied.

Message Processing

Use the exit routine, in conjunction with the installation default policy, to suppress or automate your installation's responses to the following message:

- IEF238D - Reply [device name] [,] ['wait'] or 'cancel'

  Note: In a sysplex environment, determine which, if any, devices will require job-level support; for these devices, code the exit to mark the REPLY with the device number. For all other devices (the majority), determine whether this message should set UXONLINE to 1 to allow offline devices to be brought online and to allow a pending offline device to be considered for allocation.
- IEF244I - Unable to allocate <nnn> units(s). At least <nnn> allocated or offline units are needed
- IEF433D - Wait requested — reply hold or nohold.

In addition, you might also avoid getting one or more of the following messages, which the system issues in response to invalid replies to the preceding messages:

- IEF434D - Invalid reply (to message IEF433D). Reply hold or nohold.
- IEF490I - Invalid reply (to message IEF238D) for one of the following reasons:
  - Device is not accessible
  - Required system managed volume is not available
  - Required volume is not available
  - Replied device is not eligible
  - Device could not be found in the configuration.
  - Device found in an offline library.
- IEF877E - jjobname NEEDS xxx UNIT(S) FOR stepname ddname FOR VOLUME(S): ser, ...,ser SCRTCH nn PRIVAT nn | LIBRARY: LIBNAME LIBRARY STATUS: STATUS state1 dev ...dev state2
- IEF878I - END OF IEF877E FOR stepname ddname

Macro Instructions and Restrictions

Do not code the exit routine to issue the WAIT macro or call a service that issues a WAIT, such as WTOR.

Entry Specifications

The system passes the address of the exit parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are:

Register	Contents
0	Not applicable
1	Address of a pointer to the exit parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit routine

Parameter Descriptions: Register 1 contains the address of a pointer to the exit parameter list, the UXPARMD, which is mapped by macro IEFZB481 (data area UXPARMD). For a mapping of the UXPARMD data area, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

The exit routine indicates its decision to the system by setting the ACTION field (in the UXPARMD) to one of the following values:

Value	Meaning
X'80'	Cancel the job
X'40'	Issue the WTOR so that the operator can determine what to do
X'20'	Let the job wait without holding resources
X'10'	Let the job wait while holding resources
X'08'	Bring the device online or, if pending offline, allow the device to be considered for allocation
X'00'	Let the installation default policy determine what to do

If the exit routine does not return a valid value in the ACTION field, the system will ignore the exit, issue a message, and use the installation default policy to make the decision.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents at entry
15	0

Coded Example of the Exit Routine

For your reference, IBM provides a coded example of this exit routine in SYS1.SAMPLIB. The member is named IEFOFLNE.

Chapter 18. IEF_ALLC_EVENT — Allocation Event Installation Exit

Topics for This Exit Appear as Follows:

The IEF_ALLC_EVENT exit is driven at various points in the allocation process, for example, at the start and end of batch allocation or dynamic allocation. Parameters describing the function are passed to the exit. These parameters are mapped by the IEFALCXT mapping macro.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the Allocation Event installation exit to the dynamic exits facility. You can refer to the exit by the name IEF_ALLC_EVENT. You can use the CSVDYNEX macro to control this exit and its exit routines.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error.
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Replacing the Exit Routine

For information about replacing a dynamic exit routine, see "Replacing a Dynamic Exit Routine" on page 6.

Exit Routine Environment

The exit routine receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1.
- In AMODE 31 and RMODE ANY.
- With no locks held.

Exit Recovery

The exit routine should provide its own recovery. If the exit routine abnormally terminates, its recovery routine will get control.

If the exit routine abnormally terminates, and the exit routine does not provide its own recovery, or the error percolates beyond the exit's recovery routine, a system recovery routine will get control and fail the allocation request.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

MVS invokes the Allocation Event exit routine or routines, if any are specified to the dynamic exits facility, whenever an event takes place.

Using the Information in the Parameter List: MVS passes the address of a list of parameters to the exit routine. The parameters contain the following information:

- Eye-Catcher ALCXT
- Version ID
- Length of the parameter list
- Job name
- Step name
- Name of the step in the procedure
- One of the following functions indicating the nature of the event:
  - Beginning of batch allocation – 01
  - End of batch allocation -- 02
  - Beginning of dynamic allocation – 03
  - End of dynamic allocation – 04
  - Abend in allocation -- 05
  - Concatenate DD function -- 06
  - Deconcatenate DD function -- 07
  - Unallocation (batch) -- 08
  - Unallocation (Dynamic) -- 09
- Pointer to an area, which contains function related data
  - End of batch allocation
    Return code (0 for Success and 4 for Failure)
  - Beginning of dynamic allocation request
    DDNAME
  - End of dynamic allocation request

DDNAME
Return code (0 for Success and 4 for Failure)

The exit routine can use the information in the parameter list. Because there is no action to be taken, the exit routine does not need to provide any response. MVS does not expect any response from the Allocation Event Exit.

Programming Considerations

Observe the following conventions when coding the IEF_ALLC_EVENT Exit routine:

- Code the exit routine to be reentrant.
- Do not code the exit routine to issue dynamic allocation calls.
- Do not code the exit routine if you do not need to be notified of an Allocation event.
- The exit is called every time a step allocation begins or ends. Therefore, when coding the exit routine, you should be aware that an increased path length will increase processor utilization and may degrade performance.
- Make sure that the exit routine does not get affected when new function code support is added.

Macro Instructions and Restrictions: Do not code the exit routine to invoke dynamic allocation.

Entry Specifications

The system passes the address of the exit parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	**Contents**
0	Not applicable
1	Address of a pointer to the exit parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit routine

Parameter Descriptions: Register 1 contains the address of a pointer to the exit parameter list, which is mapped by macro IEFALCXT.

Return Specifications

MVS does not provide any field for a response; the exit does not return a valid value.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	**Contents**

0-14 Restored to contents at entry

15 0

Coded Example of the Exit Routine

There is no coded example of this exit routine in SYS1.SAMPLIB.

Chapter 19. IEF_ALLC_MOD — Allocation Modify DDname Installation Exit

Topics for This Exit Appear as Follows:

The IEF_ALLC_MOD exit is driven when the service routine IEFDDSRV requests to modify a DDNAME. Parameters describing the modification are passed to the exit. These parameters are mapped by the IEFDISXT mapping macro.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the Allocation Modify DDname installation exit to the dynamic exits facility. You can refer to the exit by the name IEF_ALLC_MOD. You can use the CSVDYNEX macro to control this exit and its exit routines.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error.
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Replacing the Exit Routine

For information about replacing a dynamic exit routine, see "Replacing a Dynamic Exit Routine" on page 6.

Exit Routine Environment

The exit routine receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1.
- In AMODE 31 and RMODE ANY.
- With no locks held.

Exit Recovery

The exit routine should provide its own recovery. If the exit routine abnormally terminates, its recovery routine will get control.

If the exit routine abnormally terminates, and the exit routine does not provide its own recovery, or the error percolates beyond the exit's recovery routine, a system recovery routine will get control and fail the allocation request.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

MVS invokes the Allocation Modify exit routine or routines, if any are specified to the dynamic exits facility, every time it modifies DDname for an IEFDDSRV service request.

Using the Information in the Parameter List: MVS passes the address of a list of parameters to the exit routine. The parameters contain the following information:

- Eye-catcher (IEFDISXT)
- Version ID
- Length of the parameter list
- Modify function
  - Modify DDNAME – 01
- Job name
- Step name
- Name of step in the procedure
- Modify Parameters
  - Address of the DSAB modified
  - DDNAME before Modification
  - DDNAME after Modification

The exit routine can use the information in the parameter list. Because there is no action to be taken, the exit routine does not need to provide any response. MVS will not expect any response from the Modify Exit.

Programming Considerations

Observe the following conventions when coding the IEF_ALLC_MOD Exit routine:

- Code the exit routine to be reentrant.
- Do not code the exit routine to issue dynamic allocation calls.

- Do not code the exit routine if you do not need to be notified of a DDname Modification by the IEFDDSRV request.
- The exit is called every time a DDname is modified by IEFDDSRV. Therefore, when coding the exit routine, you should be aware that an increased path length will increase processor utilization and may degrade performance.

Macro Instructions and Restrictions: Do not code the exit routine to invoke dynamic allocation.

Entry Specifications

The system passes the address of the exit parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of a pointer to the exit parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit routine

Parameter Descriptions: Register 1 contains the address of a pointer to the exit parameter list, which is mapped by macro IEFDISXT.

Return Specifications

MVS does not provide any field for a response; the exit does not return a valid value.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents at entry
15	0

Coded Example of the Exit Routine

There is no coded example of this exit routine in SYS1.SAMPLIB.

Chapter 20. IEF_ALLC_UNLOAD — Allocation Event Installation Exit

Topics for This Exit Appear as Follows:

The IEF_ALLC_UNLOAD exit is driven when a device is unloaded by Device Allocation outside of normal end-of-job or end-of-step device unload processing. It is invoked during processing such as VARY OFFLINE or UNLOAD commands.

This exit is called before the unload event starts and after the unload is complete. It is intended for programs that need to perform pre-processing or post-processing for an unload event. If your program only needs to be notified of an unload occurring and does not need to perform pre- or post-processing, then ENF function code 25 should be used instead.

Parameters describing the function are passed to the exit. These parameters are mapped by the IEFUNLXT mapping macro.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the Allocation Event installation exit to the dynamic exits facility. You can refer to the exit by the name IEF_ALLC_UNLOAD. You can use the CSVDYNEX macro to control this exit and its exit routines.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error.
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Replacing the Exit Routine

For information about replacing a dynamic exit routine, see "Replacing a Dynamic Exit Routine" on page 6.

Exit Routine Environment

The exit routine receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1.
- In AMODE 31 and RMODE ANY.
- With no locks held.
- May be invoked with any of the following ENQs held:
  SYSIEFSD.Q4
  SYSIEFSD.VARYDEV
  SYSIEFSD.CHNGDEVS

The parameter list contains information indicating which of the ENQs are obtained by another task on behalf of the unload process.

Exit Recovery

The exit routine should provide its own recovery. If the exit routine abnormally terminates, its recovery routine will get control.

If the exit routine abnormally terminates, and the exit routine does not provide its own recovery, or the error percolates beyond the exit's recovery routine, a system recovery routine will get control and fail the allocation request.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

MVS invokes the Unload Event exit routine or routines, if any are specified to the dynamic exits facility, whenever an unload event takes place.

Using the Information in the Parameter List: MVS passes the address of a list of parameters to the exit routine. The parameters contain the following information:

- Eye-catcher (UNLXT)
- Version ID
- Length of the parameter list
- One of the following functions indicating the nature of the event:
  - Beginning of Unload event - 01
  - End of Unload event - 02
  - ABEND occurred during unload - 03
- Flags indicating which of the following ENQs are held by the caller:
  - SYSIEFSD.CHNGDEVS
  - SYSIEFSD.VARYDEV
  - SYSIEFSD.Q4
- Address of the UCB of the device being unloaded. The system does not pin the UCB before calling the exit.

The exit routine can use the information in the parameter list. Because there is no action to be taken, the exit routine does not need to provide any response from the Unload Event exit.

Programming Considerations

Observe the following conventions when coding the IEF_ALLC_UNLOAD Exit routine:

- Code the exit routine to be reentrant.
- Do not code the exit routine to issue dynamic allocation calls.
- Do not code the exit routine if you do not need to be notified of an Unload event.
- The exit is called whenever an unload event begins or ends. Therefore, when coding the exit routine, you should be aware that an increased path length will increase processor utilization and may degrade performance. This includes issuing WAITs or invoking routines that may issue WAITs. Also, because of the serialization environment, other system processes may be delayed until the exit routine completes if the exit was invoked with any ENQs held.
- Make sure that the exit routine does not get affected when new function code support is added.

Macro Instructions and Restrictions: Do not code the exit routine to invoke dynamic allocation.

Entry Specifications

The system passes the address of the exit parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the exit parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit routine

Parameter Descriptions: Register 1 contains the address of the exit parameter list, which is mapped by macro IEFUNLXT

Return Specifications

MVS does not provide any field for a response; the exit does not return a valid value.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents at entry
15	0

Coded Example of the Exit Routine

There is no coded example of this exit routine in SYS1.SAMPLIB.

Chapter 21. IEF_SPEC_WAIT — Specific Waits Installation Exit

Topics for This Exit Appear as Follows:

When a job must wait for a ***specific*** volume or device to become available, MVS issues a WTOR that requests the system operator to cancel the job or let the job wait for the volume or device. You can automate your installation's responses to specific waits allocation requests and reduce the need for system operator intervention by:

- Defining an installation default policy for handling a ***majority*** of the specific waits allocation requests that are likely to occur. Specify the default policy in the ALLOCxx member of SYS1.PARMLIB.
- Coding the specific waits exit routine to make exceptions (for certain jobs and/or volumes) to the installation default policy. You can specify the exit in the EXITxx or PROGxx member of SYS1.PARMLIB, but IBM recommends that you use PROGxx.

Using the information it receives about the job, the specific waits exit routine can:

- Cancel the job
- Allow the job to wait while:
  - holding resources
  - not holding resources
- Allow the installation default policy to determine the action to take.
- Allow the WTORs to be issued so that the system operator can determine the action to take.

For a list of the allocation messages you can automate or suppress, see the "Programming Considerations" on page 132.

For more information on the ALLOCxx, EXITxx and PROGxx members of SYS1.PARMLIB, see *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the specific waits installation exit to the dynamic exits facility. You can refer to the exit by the name IEF_SPEC_WAIT. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Replacing the Exit Routine

For information about replacing a dynamic exit routine, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

The exit routine receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1.
- In AMODE 31 and RMODE ANY.
- With no locks held. (However, it may hold an exclusive ENQ on major name SYSZTIOT for the address space in which the allocation occurs.)

Exit Recovery

The exit routine should provide its own recovery. If the exit routine abnormally terminates, its recovery routine will get control.

If the exit routine abnormally terminates, and the exit routine does not provide its own recovery, or the error percolates beyond the exit's recovery routine, a system recovery routine will get control. The system will fail the allocation request.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

MVS invokes the specific waits exit routine or routines every time a job must wait for a ***specific*** volume or a ***specific*** device to become available.

Example of a Wait for a Specific Volume: In the following example, JOB1 acquires a shareable ENQ on volume 3350A0. JOB2 attempts to acquire an exclusive ENQ on the same volume:

```
//JOB1     JOB
//DD       ...,UNIT=12A0,VOL=SER=3350A0...


//JOB2     JOB
//DD       ...,UNIT=12A1,VOL=SER=3350A0...
```

Because JOB2 must have exclusive access to the volume, JOB2 must wait for JOB1 to DEQ from the volume. MVS invokes the specific waits exit for JOB2's allocation request because JOB2 must wait for a specific volume.

Example of a Wait for a Specific Device: In the following example, JOB1 and JOB2 acquire ENQs on different specific volumes, 3350A2 and 3350B2. JOB2 specifies the same device, 2A0, that JOB1 controls.

```
//JOB1     JOB
//DD       ...,UNIT=12A0,VOL=SER=3350A2...


//JOB2     JOB
//DD       ...,UNIT=12A0,VOL=SER=3350B2...
```

MVS invokes the specific waits exit for JOB2's allocation request because JOB2 must wait for a specific device.

Using Information in the Exit Parameter List: Before issuing WTORs that identify the volume(s) that are unavailable and request operator action, MVS invokes the specific waits exit routine, if one is specified, and passes it a set of parameters that contains the following information about the allocation request:

- Job name
- Step name
- Name of the DD statement that requires the specific volume or device
- Name of the data set that requires the specific volume or device
- Serial number of the specific volume (if one is required)
- Device number of the device that is required
- An indication of whether the job is waiting for both a specific volume and a device, or for a specific device only.
- The number of 'wait without holding resources' decisions that the system will allow both the exit routine and the installation default to make for a particular device allocation request. This number (in the WAITNOHC field of the exit parameter list) does not limit the number of 'wait without holding resources' decisions that can be made by the system operator.
- Action to be taken in response to WTOR (this field, ACTION, is filled in by the exit routine).

Using the information in the parameter list, the exit routine determines how the system should respond to the allocation request. The exit routine indicates its decision to the system by placing a value in the ACTION field of the exit parameter list.

See “Return Specifications” on page 133 for the specific values the exit routine can return.

Using the Exit with Your Installation's Default Policy: If you code the exit, use it in conjunction with your installation default policy for jobs that must wait for

specific volumes. Determine your installation's default policy by specifying one of the following parameters on the POLICY parameter of the SPEC_WAIT statement:

Parameter
Action

WTOR
Allow the specific wait messages to be issued so that the system operator must decide whether to cancel the job or let the job wait. (For a list of these messages, see "Programming Considerations.")

WAITHOLD
Allow jobs to wait for devices while holding obtained resources.

WAITNOH
Allow jobs to wait for devices without holding obtained resources.

CANCEL
Cancel jobs that must wait for a volume to be released.

When you have chosen a default policy to handle the majority of specific wait WTORs that can occur, use the specific waits exit routine to make exceptions, if any, for certain jobs and/or volumes. The exit routine's decisions will override the installation's default policy.

If you do not code the specific waits exit routine, MVS will use your installation's default policy (specified in the ALLOCxx member) to determine how to respond to ***all*** specific waits allocation requests. If your installation does not define a default policy, the system will always issue the specific wait WTOR.

Programming Considerations

Observe the following conventions when coding the specific waits exit routine:

- Code the exit routine so that it is reentrant.
- Do not code the exit routine to issue dynamic allocation calls.
- Do not code the exit routine if its decision will always be the same regardless of which jobs are waiting or which volumes are needed. Instead, allow your installation's default policy to make the decision.
- The exit is called every time a job requires a specific volume. Therefore, when coding the exit routine, you should be aware that an increased path length will increase processor utilization and may degrade performance.

Message Processing: Use the exit routine, in conjunction with the installation default policy, to suppress and automate your installation's responses to the following messages:

- IEF238D - Reply [device name] [,] ['wait'] or 'cancel'
- IEF244I - Unable to allocate <nnn> units(s). At least <nnn> allocated or offline units are needed.
- IEF433D - Wait requested — reply hold or nohold
- IEF488I - Must wait for a unit, or volume on unit.

In addition, you might also avoid getting one or more of the following messages which the system issues in response to an invalid reply to the preceding messages:

- IEF434D - Invalid reply (to message IEF433D). Reply hold or nohold.
- IEF490I - Invalid reply (to message IEF238D) for one of the following reasons:
  - Device is not accessible

- Required system-managed volume is not available
- Required volume is not available
- Replied device is not eligible
- Device could not be found in the configuration.

Macro Instructions and Restrictions: Do not code the exit routine to issue the WAIT macro or call a service that issues a WAIT, such as WTOR.

Entry Specifications

The system passes the address of the exit parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of a pointer to the exit parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit routine

Parameter Descriptions: Register 1 contains the address of a pointer to the exit parameter list, the UXPARMC, which is mapped by macro IEFZB480 (data area UXPARMC). For a mapping of the UXPARMC data area, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

The exit routine indicates its decision to the system by setting the ACTION field (in the UXPARMC) to one of the following values:

Value	Meaning
X'80'	Cancel the job
X'40'	Issue the WTOR so that the operator can determine what to do
X'20'	Let the job wait without holding resources
X'10'	Let the job wait while holding resources
X'00'	Let the installation default policy determine what to do

If the exit routine does not return a valid value, the system will ignore the exit and use the installation default policy to make the decision.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents at entry
15	0

Coded Example of the Exit Routine

For your reference, IBM provides a coded example of this exit routine in SYS1.SAMPLIB. The routine is named IEFSWAIT.

Chapter 22. IEF_VOLUME_ENQ — Volume ENQ Installation Exit

Topics for This Exit Appear as Follows:

When a job must wait to enqueue on a volume or a series of volumes, MVS issues a WTOR that requests the system operator to cancel the job or let the job wait. You can automate your installation's responses to volume ENQ WTORs and reduce the need for operator intervention by:

- Defining an installation default policy for handling a ***majority*** of the volume ENQ allocation requests that are likely to occur. Specify the default policy on the VOL_ENQ statement in the ALLOCxx member of SYS1.PARMLIB.
- Coding the volume ENQ exit routine to make exceptions, if any, to the installation default policy for certain jobs and/or volumes. You can specify the exit in the EXITxx or PROGxx member of SYS1.PARMLIB; however, IBM recommends that you use PROGxx.

Using the information it receives about the job, the volume ENQ exit routine determines whether to:

- Cancel the job and suppress the WTOR
- Allow the job to wait
- Allow the WTOR to be issued so that the system operator must decide whether to cancel the job or let the job wait
- Allow the installation default policy to determine whether to cancel the job or issue the WTOR.

For a list of the allocation messages you can automate or suppress, see "Programming Considerations" on page 137.

For more information on the ALLOCxx, EXITxx, and PROGxx parmlib members, see *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the volume ENQ installation exit to the dynamic exits facility. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Replacing the Exit Routine

For information about replacing a dynamic exit routine, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

The exit routine receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1.
- In AMODE 31 and RMODE ANY.
- With no locks held. (However, it may hold an exclusive ENQ on major name SYSZTIOT for the address space in which the allocation occurs.)

Exit Recovery

The exit routine should provide its own recovery. If the exit routine abnormally terminates, its recovery routine will get control.

If the exit routine abnormally terminates, and the exit routine does not provide its own recovery, or the error percolates beyond the exit's recovery routine, a system recovery routine will get control. The system will fail the allocation request.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

MVS invokes the volume ENQ exit routine or routines every time a job must wait to enqueue on a volume. Before issuing messages (listed in “Programming Considerations” on page 137) that identify the volume(s) and request operator action, MVS invokes the volume ENQ exit routine or routines, if any are specified to the dynamic exits facility, and passes a list of parameters that contains the following information about the allocation request:

- Job name

- Step name
- A table containing the serial numbers of the volumes that the job requires
- Action to be taken in response to WTOR (this field, ACTION, is filled in by the exit routine).

Using the information in the parameter list, the exit routine indicates to the system (in the ACTION field of the parameter list) whether the system should:

- Cancel the job and suppress the WTORs
- Allow the job to wait for the volume(s)
- Allow the WTORs to be issued so that the system operator must make the decision
- Allow the installation's default policy to make the decision.

See “Return Specifications” on page 138 for the specific values the exit routine can return.

Using the Exit with Your Installation's Default Policy: If you code the exit, use it in conjunction with your installation default policy for jobs that must wait to enqueue on volumes. Determine your installation's default policy by specifying one of the following parameters on the POLICY parameter of the VOLUME_ENQ statement:

Parameter
Action

CANCEL
Cancel jobs that must wait for a volume to be released

WTOR
Allow the volume ENQ messages to be issued so that the system operator must decide whether to cancel the job or let the job wait.

WAIT Allow jobs to wait for volumes to be released.

Attention: When WAIT is used as the default, deadlocks with other jobs in the system might arise for tape volumes.

When you have chosen a default policy to handle the majority of volume ENQ WTORs that can occur, use the volume ENQ exit routine to make exceptions, if any, for certain jobs and/or volumes. The exit routine's decisions will override the installation's default policy.

If you do not code the volume ENQ exit routine, MVS will use your installation's default policy (specified in the ALLOCxx member) to determine how to respond to ***all*** volume ENQ allocation requests. If your installation does not define a default policy, the system will always issue the volume ENQ WTORs.

Programming Considerations

Observe the following conventions when coding the volume ENQ exit routine:

- Code the exit routine so that it is reentrant.
- Do not code the exit routine to issue dynamic allocation calls.
- Do not code the exit routine if its decision will always be the same regardless of which jobs are waiting or which volumes are needed. Instead, allow your installation's default policy to make the decision.

- The exit is called every time a job requires a volume to be released. Therefore, when coding the exit routine, you should be aware that an increased path length will increase processor utilization and may degrade performance.

Message Processing: Use the exit routine, in conjunction with the installation default policy, to suppress and automate your installation's responses to the following messages:

- IEF690I - The following volumes are unavailable to <jobname>...
- IEF235D - <jobname> is waiting for volumes. To cancel wait, reply no.

In addition, you might avoid getting message IEF369D (invalid reply), which the system issues in response to an invalid reply to IEF235D.

Macro Instructions and Restrictions: Do not code the exit routine to issue the WAIT macro or call a service that issues a WAIT, such as WTOR.

Entry Specifications

The system passes the address of the exit parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of a pointer to the exit parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit routine

Parameter Descriptions: Register 1 contains the address of a pointer to the exit parameter list, the UXPARMA, which is mapped by macro IEFZB478 (data area UXPARMA). For a mapping of the UXPARMA data area, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

The volume ENQ exit indicates its decision to the system by placing one of the following values in the 1-byte ACTION field of the exit parameter list (the UXPARMA):

Value	Explanation
X'80'	Cancel the job and suppress the WTOR
X'40'	Issue the WTOR so that the system operator can make the decision
X'08'	Let the job wait for the volume(s)
X'00'	Let the installation default policy make the decision

If the exit routine does not return a valid value in the ACTION field, the system will ignore the exit and use the installation default policy to make the decision.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents at entry
15	0

Coded Example of the Exit Routine

For your reference, IBM provides a coded example of this exit routine in SYS1.SAMPLIB. The routine is named IEFVENQS.

Chapter 23. IEF_VOLUME_MNT — Volume Mount Installation Exit

Topics for This Exit Appear as Follows:

When a job's allocation request requires a volume to be mounted, MVS issues a WTOR that requests the system operator to mount the volume or cancel the job. You can automate your installation's responses to volume mount WTORs, and reduce the need for operator intervention, by defining an installation default policy for volume mount allocation requests in the ALLOCxx member of SYS1.PARMLIB. The policy you specify will handle the majority of volume mount allocation requests that can occur.

Code a volume mount exit routine when you want to make exceptions to your installation default policy for certain jobs and/or volumes. Using the information it receives about the job, the exit routine determines whether to:

- Cancel the job and suppress the WTOR
- Allow the WTOR to be issued so that the system operator must decide whether to cancel the job or mount the volume
- Allow your installation's default policy to determine whether to cancel the job or issue the WTOR.

For a list of the allocation messages you can automate or suppress, see "Programming Considerations" on page 143.

For more information on the ALLOCxx and EXITxx members of SYS1.PARMLIB, see *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the volume mount exit to the dynamic exits facility. You can refer to the exit by the name IEF_VOLUME_MNT. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Replacing the Exit Routine

For information about replacing a dynamic exit routine, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

The exit receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1.
- In AMODE 31 and RMODE ANY.
- With no locks held. (However, it may hold an exclusive ENQ on major name SYSZTIOT for the address space in which the allocation occurs.)
- Can reside in (E)PLPA, (E)MLPA, or (E)FLPA.

Exit Recovery

The exit routine should provide its own recovery. If the exit routine abnormally terminates, its recovery routine will get control.

If the exit routine abnormally terminates, and the exit routine does not provide its own recovery, or the error percolates beyond the exit's recovery routine, a system recovery routine will get control. The system will fail the allocation request.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility. However, the exit routine will continue to be invoked for all other volume mount allocation requests.

Exit Routine Processing

MVS invokes the volume mount exit routine or routines every time a job must wait for a volume to be mounted. Before issuing WTORs that identify the volume(s) to be mounted and request operator action, MVS invokes the volume mount exit routine or routines, if any are specified to the dynamic exits facility, and passes a set of parameters that contains the following information about the allocation request:

- Job name
- Step name
- DD name of the data set that requires the volume to be mounted
- Data set name
- Volume serial number

- Device number of the device on which the volume is to be mounted
- Flags indicating the type of label on the volume, if any
- Action to be taken in response to WTOR (this field is filled in by the exit routine).

Using the information in the parameter list, the exit routine indicates to the system (by placing a value in the ACTION field of the parameter list) whether the system should:

- Cancel the job
- Allow the WTORs to be issued so that the system operator must make the decision
- Allow the installation's default policy to make the decision.

See "Return Specifications" on page 144 for the specific values the exit routine can return.

Using the Exit with Your Installation's Default Policy: If you code the exit, use it in conjunction with your installation's default policy for jobs that require volumes to be mounted. Determine your installation's default policy by specifying one of the following parameters on the POLICY parameter of the VOLUME_MNT statement:

Parameter
: **Action**

CANCEL
: Cancel jobs that must wait for a volume to be mounted.

WTOR
: Allow the volume mount messages to be issued so that the system operator must decide whether to cancel the job or mount the volume.

When you have chosen a default policy to handle the majority of volume mount WTORs that can occur, use the volume mount exit routine to make exceptions, if any, for certain jobs and/or volumes. The exit routine's decisions will override the installation's default policy.

If you do not code the volume mount exit routine, MVS will use your installation's default policy (specified on the ALLOCxx parmlib member) to determine how to respond to ***all*** allocation volume mount requests (either cancelling all jobs that must wait for a volume to be mounted, or always allowing the WTOR to be issued). If your installation does not define a default policy, the system will always issue the volume mount WTOR.

Programming Considerations

Observe the following conventions when coding the volume mount exit routine:

- Code the exit routine to be reentrant.
- Do not code the exit routine to issue dynamic allocation calls.
- Do not code the exit routine if its decision will always be the same regardless of which volumes are needed. Instead, allow your installation's default policy to make the decision.
- The exit is called every time a job requires a volume to be mounted. Therefore, when coding the exit routine, you should be aware that an increased path length will increase processor utilization and may degrade performance.

Message Processing: Use the exit routine, in conjunction with the installation default policy, to suppress and automate your installation's responses to the following messages:

- IEF233A - Mount volume <ser>
- IEF233D - Mount volume <ser> or respond to IEF455D message
- IEF455D - Mount <ser> on <device> for <jobname> or reply no.

In addition, you might avoid getting message IEF369D (invalid reply), which the system issues in response to an invalid reply to IEF455D.

Macro Instructions and Restrictions: Do not code the exit routine to issue the WAIT macro or call a service that issues a WAIT, such as WTOR.

Entry Specifications

The system passes the address of the exit parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of a pointer to the exit parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit

Parameter Descriptions: Register 1 contains the address of a pointer to the exit parameter list, the UXPARMB, which is mapped by macro IEFZB479 (data area UXPARMB). For a mapping of the UXPARMB data area, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

The volume mount exit indicates its decision to the system by placing one of the following values in the 1-byte ACTION field of the exit parameter list:

Value	Explanation
X'80'	Cancel the job and suppress the WTOR
X'40'	Issue the WTOR so that the system operator can make the decision
X'00'	Let the installation default policy make the decision

If the exit routine does not return a valid value, the system will ignore the exit and use the installation default policy to make the decision.

If you associate multiple exit routines with IEF_VOLUME_MNT, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents at entry
15	0

Coded Example of the Exit Routine

For your reference, IBM provides a coded example of this exit routine in SYS1.SAMPLIB. The routine is named IEFVOLMT.

Chapter 24. IEFACTRT — SMF Job and Job Step Termination Exits

Topics for This Exit Appear as Follows:

IEFACTRT receives control from the system when a job or job step terminates, either normally or abnormally. A return code from IEFACTRT (in register 15) indicates whether the job is to continue or terminate. Another return code (in register 1) indicates whether or not SMF is to write the termination records to the SMF data set.

The system invokes IEFACTRT only when the installation is collecting SMF record types 4, 5, 30, 32, 34, or 35. IEFACTRT is invoked for:

- Record types 4, 5, and 30 for background jobs and started tasks.
- Record types 30, 32, 34, and 35 for TSO/E users.
- Record type 30 (subtypes 4 and 5) for work initiated by the IBM-supplied APPC/MVS transaction scheduler (ASCH).

When the data for an SMF record exceeds 32,756 bytes in length, the system constructs one or more "continuation" or "additional" records to ensure that no individual record exceeds that length. The system invokes IEFACTRT once for the original record and once for each continuation record.

The system invokes IEFACTRT for these types of records even when an installation uses the SYS(NOTYPE) parameter in SMFPRMxx to suppress a particular subtype of these record types. Only when the installation suppresses an entire record type through SMFPRMxx will the system not pass the record to IEFACTRT.

The system does not invoke IEFACTRT for any other record types.

You can use IEFACTRT to:

- Write selected job or job step records to an installation-defined data set for further analysis.
- Displacement from PointerInclude additional information in the SMF job/job step termination records.
- Write messages to a job log to provide additional information about the job or job step. For example, if the operator has cancelled a job, you can issue a WTOR to learn why the job was cancelled and then write the reason as a message to the job log.
- Write an estimated job or job step cost to the job log.
- Update tables that describe the amount of resources certain users consume. For example, you can keep a total of the processor time for specific users, then flag their account numbers if they exceed an allowed time limit.

Defining the Exit in SMFPRMxx

In the SMF parmlib member (SMFPRMxx), specify IEFACTRT on the EXITS option of the SYS or SUBSYS parameters, depending on the scope of work (system-wide or subsystem-wide) the exit is to affect.

If you use the SUBSYS option, the system invokes the IEFACTRT routine only for work running under the subsystems you specify on SUBSYS. If you use the SYS option, the system invokes the IEFACTRT routine for work running under any SMF-defined subsystem, such as JES2, JES3, STC, ASCH, OMVS, or TSO.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFACTRT installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFACTRT or SYSyyy.IEFACTRT. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

To define IEFACTRT to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for IEFACTRT, you need only define this exit in SMFPRMxx.

If you do not associate any exit routines with exit IEFACTRT in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFACTRT).

If you associate exit routines with IEFACTRT in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFACTRT receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31.
- In the address space of the task that is currently running.
- With no locks or ENQs held.

Exit Recovery

IBM strongly recommends that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of IEFACTRT.

An ESTAE-type recovery routine is set up by the module that calls IEFACTRT; the recovery routine, if it gets control, will allow the job to continue processing if the exit routine abnormally ends.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

SMF constructs several types of termination records. An installation can use the IEFACTRT exit routine to include additional information in the termination record and to determine whether to write the termination record to the SMF data set. The specific SMF termination records that IEFACTRT references are:

Dec (Hex)
Description

4 (X'04')
The step termination record is constructed and written at the normal or abnormal termination of a job or job step for a background job, or when a job step is flushed during or after job initiation.

5 (X'05')
The job termination record is constructed and written at the normal or abnormal termination of a background job.

30 (X'1E')
The common address space work record is written at the normal or abnormal termination of a batch job or step, a TSO/E session, a started task, or work initiated by the IBM-supplied APPC/MVS transaction scheduler and at the expiration of a specified interval. The type 30 record consolidates data that also is found in record types 4, 5, 20, 34, 35, and 40. Record subtypes are used within the type 30 record to help limit the amount of data stored in SMF data sets.

32 (X'20')
The TSO/E user work accounting record is written when a TSO/E session terminates (normally or abnormally) and when a TSO/E accounting interval expires.

34 (X'22')
The TSO-step termination record is constructed and written when the TSO/E logoff function processes a job step termination.

35 (X'23')
The logoff record is constructed and written when a logoff process is completed.

At job or job step termination, use the termination indicators in record types 4, 5 and 30 to determine whether or not IEFACTRT cancelled the job.

The length of the type 30 record is variable. If the data exceeds 32,756 bytes, the system constructs "continuation" (or "additional") type 30 records. Each such record always contains these three sections:

- Header section
- Subsystem section
- Identification section

In addition, each continuation record contains one or more of the following sections:

- Execute channel program (EXCP) section
- Usage section
- Automatic restart management section
- z/OS UNIX process section

Your exit routine should be aware of these added type 30 continuation records, because they are passed to IEFACTRT. See the CAUTION bullet in the next section , "Programming Considerations," if the exit routine suppresses some of these records. See *z/OS MVS System Management Facilities (SMF)* for more information concerning type 30 records.

IEFACTRT is the only SMF exit routine that can write to a job log, but only by passing the message to module IEFYS. When IEFYS receives control, register 13 must contain the address of an 18-word save area and register 12 must be used to pass message information. Figure 5 shows the procedure for writing JOBLOG messages from IEFACTRT. The maximum number of characters that can appear in a message is 132.

```
        MVC   36(4,12),MSGADDR  MOVE MESSAGE ADDRESS AND
        MVC   42(2,12),MSGLEN   LENGTH TO SYSTEM TABLE
        L     REG15,VIEFYS      BRANCH AND LINK TO MESSAGE
        BALR  REG14,REG15       ROUTINE

MSGADDR  DC    A(MSG)
MSG      DC    C'message text'
MSGLEN   DC    H'xx'             MESSAGE LENGTH
VIEFYS   DC    V(IEFYS)
```

Figure 5. Writing Job Log Messages from IEFACTRT

To resolve the VCON for IEFYS correctly, be sure to follow the example in Figure 6 on page 151 which shows how to make an IEFACTRT installation exit routine

available to the system by link editing it into a system library.

```
//LKUSRPGM     JOB     MSGLEVEL=(1,1)
//            EXEC    PGM=IEWL,PARM='XREF,LET,LIST'
//SYSPRINT     DD      SYSOUT=A
//SYSUT1       DD      UNIT=SYSDA,SPACE=(TRK,10)
//SYSLMOD      DD      DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSLIB       DD      DSNAME=SYS1.AOSB3,DISP=SHR
//SYSLIN       DD      *
    object deck
         INCLUDE    SYSLIB(IEFTB724)
         NAME       EXITNAME(R)
/*
```

Figure 6. Example: make an IEFACTRT installation exit routine available

Programming Considerations

SMF provides a replaceable module for an unused exit. If an installation includes IEFACTRT, it must follow certain programming standards.

- The exit routine must follow standard linkage conventions. For example, upon exit, register 15 must contain the return code. (But see the note in the topic "Registers at Exit" later in this chapter .)
- Code the exit routine to be reentrant.
- IEFACTRT can perform dynamic allocations and write to installation-defined data sets. In foreground jobs, data sets are allocated dynamically. However, for background jobs, you can either allocate data sets dynamically or you can pre-define (pre-allocate) a data set with a DD statement in the initiator cataloged procedure.
- IEFACTRT cannot access ISAM data sets.
- Do not use a WTO with a routing code of 11 to send a message to the JESYSMSG data set for started tasks or TSO users.
- To provide a consistent environment for accessing and allocating data sets across calls to SMF exits for the duration of a job or task, IEFACTRT receives control with the initiator's JSCB active.
- When there are multiple data records, the IEFACTRT exit routine receives control once for each record.
- **CAUTION:** If the IEFACTRT installation exit elects to suppress one or more BUT NOT ALL of the SMF type 30 continuation records, then follow-on batch jobs that process SMF records might encounter unexpected and invalid conditions, such as:
  - "Continuation" type 30 records without the initial type 30 record
  - An initial type 30 record indicating that more type 30 records follow, but some or all of those continuation records are not present.

Macro Instructions and Restrictions: Your IEFACTRT exit routine can issue MVS system macros. Observe the following restrictions:

- Do not code your IEFACTRT exit routine to issue the WAIT macro, or call a service that issues WAIT. Doing so in IEFACTRT can adversely affect the system's allocation and unallocation functions.
- Do not use a WTO with a routing code of 11 to send a message to the JES joblog for started tasks or TSO users.

The following SMF macros are available to SMF installation exit routines:

- IFASMFR — to address SMF record fields
- SMFWTM — to write records to the SMF data set

- SMFEWTM — to write records to the SMF data set
- SMFRTEST — to test record recording
- SMFEXIT — to branch to the SMF exits
- SMFINTVL — to determine interval time
- SMFDETAL — to test detail recording
- SMFSUBP — to determine subsystem parameters
- SMFCHSUB — to change subsystem parameters.

For information on how to use these macros, see *z/OS MVS System Management Facilities (SMF)*.

Entry Specifications

SMF passes to IEFACTRT a code to define the reason for calling the exit routine and a list of parameter addresses that the routine can use.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register
Contents

0 One of the following hexadecimal codes that indicate why SMF has invoked the exit:

Code	Explanation
X'0C'	Indicates job step termination. Word 10 in the parameter list is the address of the record descriptor word (RDW) for a type 4 or type 34 record.
X'10'	Indicates job termination. Word 10 in the parameter list is the address of the RDW for a type 5 or type 35 record.
X'14'	Indicates job or step termination. Word 10 in the parameter list is the address of the RDW for a type 30 record. The subtype field in the type 30 record determines if it is a job or a step termination record.
X'18'	Indicates TSO/E session or accounting interval termination. Word 10 in the parameter list is the address of the RDW for a type 32 record.

1 Address of the parameter list

2-11 Not applicable

12 Address of a message information area, which can be used to interface to IEFYS.

13 Register save area

14 Return address

15 Entry point address of IEFACTRT

Parameter Descriptions: Register 1 points to the following list of addresses:

Word 1
Address of the common exit parameter area. For details, see “Common Exit Parameter Area” on page 155.

Word 2

Address of an 8-byte area containing the job step name (in EBCDIC). This area is aligned left and padded with blanks if necessary. At job termination, the field is zero.

Note: It is possible that some address spaces will be associated with IEESYSAS. For that case, JMRJOB will contain IEESYSAS.

Word 3

Address of a 20-byte area containing the programmer's name (in EBCDIC). This area is aligned left and padded with blanks if necessary.

Word 4

Address of a 4-byte area. The first 3 bytes contain the job processor time (which includes time under TCBs, enclave time, preemptable class SRB time, and client SRB time), in hundredths of a second (in binary). The last byte contains the number of accounting fields in the JOB statement. At job termination, the field is zero.

Note: Since this job processor time is only 3 bytes it is limited to 46 hours. If the job accumulates more than this amount of processor time, this field will become invalid. You can avoid this situation by having your exit get the processor time from the parameter at word 11.

Word 5

Address of a variable length area that contains accounting fields from the JOB statement. For details, see Table 4 on page 157.

Word 6

Address of a 4-byte area. The first 3 bytes contain the step processor time (which includes time under TCBs, enclave time, preemptable class SRB time, and client SRB time), in hundredths of a second (in binary). The last byte contains the number of accounting fields in the EXEC statement (in binary). At job termination, the field is zero.

Note: Since this step processor time is only 3 bytes it is limited to 46 hours. If the step accumulates more than this amount of processor time, this field will become invalid. You can avoid this situation by having your exit get the processor time from the parameter at word 12.

Word 7

Address of a variable length area that contains accounting fields from the EXEC statement. For details, see Table 4 on page 157.

Word 8

Address of a 2-byte area. The first byte is a binary indicator; if bit 7 (low-order bit) is set to 1 when the exit routine is entered, the job has been cancelled. If the exit routine sets bit 7 to 1, the job will be cancelled. The second byte contains the number of the job step currently being processed. At job termination, the second byte contains the number of steps in the job.

Word 9

Address of a 2-byte area containing the termination status (condition or completion code) for the job or job step. For information on the job or job step termination status, see the description of the following fields in *z/OS MVS System Management Facilities (SMF)*:

- Record Type 4 — field SMF4SCC
- Record Type 5 — field SMF5JCC
- Record Type 30 — field SMF30SCC

- Record Type 34 — field TIVSTAT

Word 10

Address of an area containing a 4-byte record descriptor word (RDW). One of the following records immediately follows the RDW:

- The job step termination record (type 4 or type 34)
- The job termination record (type 5 or type 35)
- The common address space work record (type 30)
- The TSO/E command accounting record (type 32)

Word 11

Address of a 4-byte area containing the job processor time, which includes time under TCBs, enclave time, preemptable class SRB time, and client SRB time, in hundredths of a secondary (in binary).

Word 12

Address of a 4-byte area containing the step processor time, which includes time under TCBs, enclave time, preemptable class SRB time, and client SRB time, in hundredths of a second (in binary). At job termination, the field is zero.

Word 13

Address of a 4-character area that contains the name of the subsystem for the job being processed. Examples:

- ASCH, JES2, or JES3 - indicates the name of the subsystem that selected the job
- OMVS - indicates a forked or spawned address space
- STC - indicates a started task
- TSO - indicates a time sharing option task
- The jobname - used if it is four or fewer characters and none of the above apply

Note: The high-order bit is set in the address of the last parameter to indicate the end of the parameter list.

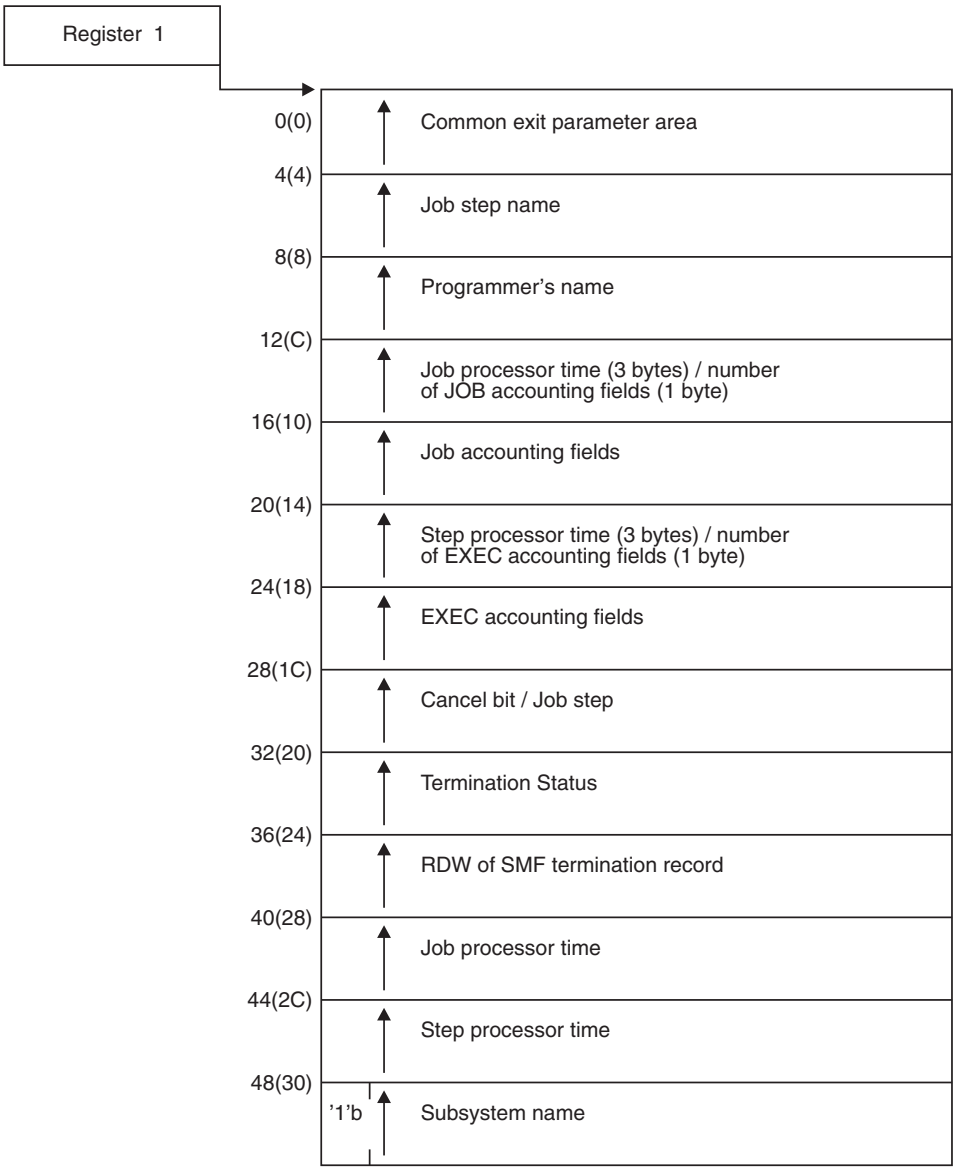

Figure 7. IEFACTRT Input Parameter Structure

Common Exit Parameter Area

The common exit parameter area is a 56-byte area that contains information an SMF installation exit routine might need. It is a copy of the first 56 bytes of the job management record (JMR), of which the first 36 bytes are usable by the exit. In addition, byte 56 is a version indicator, which indicates whether additional information is available. When byte 56 is '01'x, an additional area is provided at offset 76, mapped by the JMRE DSECT within the JMR and contains an 8 character jobclass and a 64-byte job correlator, if provided by the primary JES subsystem.

The address of the common exit parameter area is passed to all SMF installation exits except IEFU29, IEFU83, IEFU84, and IEFU85. The common exit parameter area is mapped by macro IEFJMR, as part of the JMR, except for the indicator of the SMF option selected by the user field. This field is mapped by the SMCAOPT field in the SMCA data area. See Table 3 on page 156 for a description of the area, and "Accounting Information" on page 157 for a description of the accounting

information. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the complete mapping of the JMR and the complete mapping of the SMCA.

Table 3. Common Exit Parameter Area

Displacement from Pointer	Field Size	Data Format	Description
0	8	EBCDIC	Job name
8	4	binary	Time, in hundredths of a second, when the reader recognized the JOB statement for this job
12	4	packed	Date when the reader recognized the JOB statement for the job, in the form 0CYYDDDF where F is the sign and C is X '1' if the year is greater than 1999
16	4	EBCDIC	System identification (taken from SID parameter)
20	8	EBCDIC	User identification. SMF places this data in all subsequent records for this job. This field is initialized to EBCDIC blanks when each job is read. **Note**: This field is not related to the USER parameter on the JOB statement.
28	1	binary	Number of the step being processed
29	1	binary	Indicator of the SMF options selected by the user. (The following bit settings are mapped by the SMCAOPT field in the SMCA data area. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for a mapping of the SMCA data area.) **Bit** **Meaning When Set** **0** Reserved **1** Reserved **2** Reserved **3** Data set accounting.[1] **4** Volume accounting. Record type 19 selected **5** Usage data collection services **6** Type 17 records will be written for temporary data sets (REC(ALL)) **7** If 0, background job. If 1, foreground job. **Note:** TSO sessions, APPC/MVS transactions, and OMVS (z/OS UNIX System Services) forked and spawned jobs are indicated as foreground jobs.
30	1	binary	Restart indicator **Bit** **Meaning When Set** **0** Automatic step restart **1** Automatic checkpoint/restart **2** Continue restart **3** Reserved **4** Warm start **5-7** Reserved
31	1	EBCDIC	Job class

Table 3. Common Exit Parameter Area (continued)

Displacement from Pointer	Field Size	Data Format	Description
32	4	binary	The user-communication field. This field is intended for communication among user-written exit routines within a unique job. The field is initialized to zeroes when a job begins execution. For APPC/MVS transactions, this field remains zeros.
36	19	binary	Information not usable to the exit
55	1	binary	Version of JMR
56	20	binary	Information not usable to the exit
76	8	EBCDIC	Job class
84	64	EBCDIC	Job correlator, if provided by primary Job subsystem
Note: [1] This bit is set on when any of the following record types is selected: 14, 15, 17, 18, 62, 63, 64, 67, or 68.			

Accounting Information

Accounting Information: These SMF installation exits receive accounting information for the job or job step:

- IEFACTRT
- IEFUAV
- IEFUJI
- IEFUSI

The accounting information is presented to IEFACTRT differently than to IEFUAV, IEFUJI, and IEFUSI. IEFACTRT receives both the number of accounting fields and a pointer to the accounting fields in the input parameter list. See the parameter descriptions in "Entry Specifications" on page 152 for more details. For the other exits, the input parameter list contains a pointer to all the accounting information as described in Table 4.

Table 4. Format of Accounting Information

Offset	Length	Format	Description
0	1-Byte	Binary	Number of accounting fields.
1	Variable	EBCDIC	Accounting fields. Each accounting field contains the length of the field (one byte, binary) followed by accounting information (variable length, EBCDIC). A zero in the length field indicates an omitted field.

Figure 8 on page 158 provides examples of accounting information. Note that these fields are contiguous and are only spaced in Figure 8 on page 158 for clarity. These examples show different forms of accounting information you can specify in JCL, and the resulting representation (hexadecimal) that will be passed to the SMF exits listed earlier .

In the examples in Figure 8 on page 158:

- 'MYJOB' represents the job name.
- 'PROGNAM' represents the programmer name.
- '#FIELDS' represents the number of accounting fields.

- 'LEN1', 'LEN2', and 'LEN3' represent the length of the respective accounting field.
- 'DATA1', 'DATA2', and 'DATA3' represent data in the respective accounting field.

```
                                .........Accounting Information.........
   JCL                                  .......Accounting Fields........
                                #FIELDS LEN1 DATA1 LEN2 DATA2 LEN3 DATA3
-------------------------------------------------------------------------
//MYJOB JOB ,PROGNAM          -> 01      00
//MYJOB JOB 9,PROGNAM         -> 01      01   F9
//MYJOB JOB (9,8),PROGNAM     -> 02      01   F9    01   F8
//MYJOB JOB (9,8,77),PROGNAM  -> 03      01   F9    01   F8    02   F7F7
//MYJOB JOB '9',PROGNAM       -> 01      01   F9
//MYJOB JOB '9,8',PROGNAM     -> 01      03   F96BE8
//MYJOB JOB ('9','8'),PROGNAM -> 02      01   F9    01   F8
//MYJOB JOB ('9,8'),PROGNAM   -> 01      03   F96BF8
//MYJOB JOB (9,,8),PROGNAM    -> 03      01   F9    00         01   F8
//MYJOB JOB (9,',8),PROGNAM   -> 03      01   F9    00         01   F8
//MYJOB JOB ((9)),PROGNAM     -> 01      01   F9
```

Figure 8. Examples of Accounting Information

Return Specifications

A return code from IEFACTRT (in register 15) indicates whether the job is to continue or terminate. Another return code (in register 1) indicates whether or not SMF is to write the termination records to the SMF data set.

If you associate multiple exit routines with IEFACTRT, you can use any of these criteria methods to specify how the system is to handle the return information:

- The ATTRIB KEEPRC function of the SETPROG EXIT command,
- The ATTRIB KEEPRC parameter of the EXIT statement of PROGxx, or
- The RCFROM/RCCOMPARE parameters of the CSVDYNEX services.

The criterion is based on the contents of register 15. If multiple exit routines match the defined criterion, the system returns information from the exit routine called first.

```
The defined criterion is EXIT ATTRIB KEEPRC(GE,4).

If  Exit A returns with  R1 = 4  and  R15 = 0,
and Exit B returns with  R1 = 0  and  R15 = 8,
and Exit C returns with  R1 = 0  and  R15 = 4,
and Exit D returns with  R1 = 4  and  R15 = 4,

the final results are    R1 = 0  and  R15 = 8.

The system derives the return information from the first exit called that
matches the defined criteria. Here, R15 is greater or equal to 4, which
is Exit B.

The job continues (because R15 is not 4) and the system writes the
termination records to the SMF data set (because R1 for Exit B is not 4).
```

Figure 9. Example

If you do not specify a criteria method, the system returns the information from the first exit routine that returns a value of four in register 15, and cancels the remaining job steps. If that exit routine also returns a value of four in register 1, the system does not write the termination records to the SMF data set. For any other value in register 1 the system does write the termination records.

```
There are no defined criteria.

If  Exit A returns with  R1 = 4  and  R15 = 0,
and Exit B returns with  R1 = 0  and  R15 = 8,
and Exit C returns with  R1 = 0  and  R15 = 4,
and Exit D returns with  R1 = 4  and  R15 = 4,

the final results are    R1 = 0  and  R15 = 4.

The system derives the return information from the first exit called that
returns a value of 4 in R15, which is Exit C.

The job terminates (because R15 is 4) and the system writes the
termination records to the SMF data set (because R1 for Exit C is not 4).
```

Figure 10. Example

If no exit routine returns a value of four in register 15, the system returns the information from the exit routine that is called first, and the job continues. Again, if that exit routine returns a value of four in register 1, the system does not write the termination records to the SMF data set, while for any other value in register 1 from that exit routine, the system does write the termination records to the SMF data set.

```
There are no defined criteria.

If  Exit A returns with  R1 = 4  and  R15 = 0,
and Exit B returns with  R1 = 0  and  R15 = 8,
and Exit C returns with  R1 = 0  and  R15 = 0,

the final results are    R1 = 4  and  R15 = 0.

Because no exit set R15 to 4, the system derives the return information
from the first exit called, which is Exit A.

The job continues (because R15 is not 4) and the system does not write
the termination records to the SMF data set (because R1 for Exit A is 4).
```

Figure 11. Example

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0 Restored to contents at entry

1 One of the following return codes:

Return Code
Explanation

Value of 4
SMF is not to write the termination record to the SMF data set.

Other than 4
SMF is to write the termination record to the SMF data set.

2-14 Restored to contents at entry

15 One of the following return codes:

Return Code
Explanation

Value of 4
The remaining job steps are to be cancelled.

Other than 4
Job processing is to continue.

Note: The system may fail a step or job even if the return code is zero. This could happen, for example, as a result of specifying CATLG_ERR FAILJOB(YES) and incurring that type of post execution error. (A return code is generated by the application program and is never changed by the operating system.) A user can deduce that a step failed due to a "post execution error" if bit SMF30SYE in the two-byte SMF30STI field in the SMF30 subtype 4 record is on.

Coded Example of the Exit Routine

Sample IEFACTRT exit routines are provided in SYS1.SAMPLIB in members SMFEXITS and IEEACTRT. The sample in SMFEXITS changes the SMF job termination (types 5 and 35) and job step termination (types 4 and 34) records to user records, and attempts to write them to an SMF data set. If the data set is full, the routine writes a message to the console indicating that SMF records are being lost. At job termination, the routine writes a record containing the job name, programmer's name, and account number to the JOBLOG data set.

The IEEACTRT exit routine puts a summary of the step on each JES2 job log using WTO with ROUTCODE=14. The summary includes both step and job information.

Chapter 25. IEFDB401 — Dynamic Allocation Input Validation Routine Exit

Topics for This Exit Appear as Follows:

The IEFDB401 installation exit from the allocation control routine allows an exit routine to either validate or alter any dynamic allocation request. Control passes to IEFDB401 for all system and user dynamic allocation requests. See *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about dynamic allocation.

The user validation routine tests and can modify the dynamic allocation request, and it indicates through a return code whether or not processing of the request is to continue. You can use IEFDB401 to:

- Control the amount of direct access space requested.
- Check for authorization to use specified units.
- Check for authorization to use certain data sets.
- Check for authorization to hold certain resources for reuse.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFDB401 installation exit to the dynamic exits facility. You can refer to the exit by the name IEFDB401. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

If you do not associate any exit routines with exit IEFDB401 in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFDB401).

If you associate exit routines with IEFDB401 in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the

SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFDB401 receives control in the following environment:

- In supervisor state and under the scheduler's PSW key (key 1).
- In AMODE 31 and RMODE ANY.

Exit Recovery

An ESTAE routine that is established in the calling module provides recovery from errors encountered in IEFDB401.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

The dynamic allocation facility passes control to IEFDB401 before doing any processing on behalf of a dynamic allocation request. It is entered for all requests, foreground and background. IEFDB401 can test and modify the dynamic allocation input, and indicate with a return code whether processing is to continue or if the request is to be terminated. The exit receives control after System Allocation has completed validation of the passed parameters.

The IBM-supplied version of IEFDB401 that your routine can replace does no testing and allows all requests to continue processing.

Programming Considerations

- When IEFDB401 receives control, a parameter list is passed via register 1. Among the parameters are the dynamic allocation request block and a work area for the addition or modification of text units. This work area immediately follows the text unit pointer list and is usually in nonfetch-protected scheduler key storage (subpool 230). It is in fetch-protected scheduler key storage (subpool 229) only when the request has a password specification text unit. If text unit pointers are to be added to the pointer list, they must be added to the end of the list in the work area. The end-of-list indicator also must be adjusted. To delete a text unit pointer, zero the text unit pointer or the text unit key.
- Code the exit routine so that it is reentrant.

Entry Specifications

IEFDB401 receives control from dynamic allocation.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEFDB401

Parameter List Contents: Register 1 points to the following parameters:

Word 1
Address of a copy of the dynamic allocation input parameter list in scheduler-key storage (mapped by macro IEFZB4D0). See the chapter on dynamic allocation in *z/OS MVS Programming: Authorized Assembler Services Guide*. See S99PARMS in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the structure of IEFZB4D0.

Word 2
Pointer to the address of a work area that the exit routine can use. This area is contiguous with the text unit pointer list so that you can use it to extend the list and provide additional text units.

Word 3
Address of a fullword that contains the length of the work area (500 bytes).

Word 4
Address of the 8-character job name.

Word 5
Address of the 20-character programmer name.

Word 6
Address of an area that contains accounting information from the JOB statement. The first byte of this area contains the number of accounting fields; the accounting fields follow this byte. Each entry for an accounting field contains the length of the field (one byte, hexadecimal), followed by the field itself. The entry for a null field contains a length of zero.

Word 7
Address of the 8-character step name.

Word 8
Address of the 8-character program name.

Word 9
Address of an area containing accounting information from the EXEC statement. The first byte of this area contains the number of accounting fields (zero for no fields); the accounting fields follow this byte. Each entry for an accounting field contains the length of the field (one byte, hexadecimal), followed by the field itself. The entry for a null field contains a length of zero.

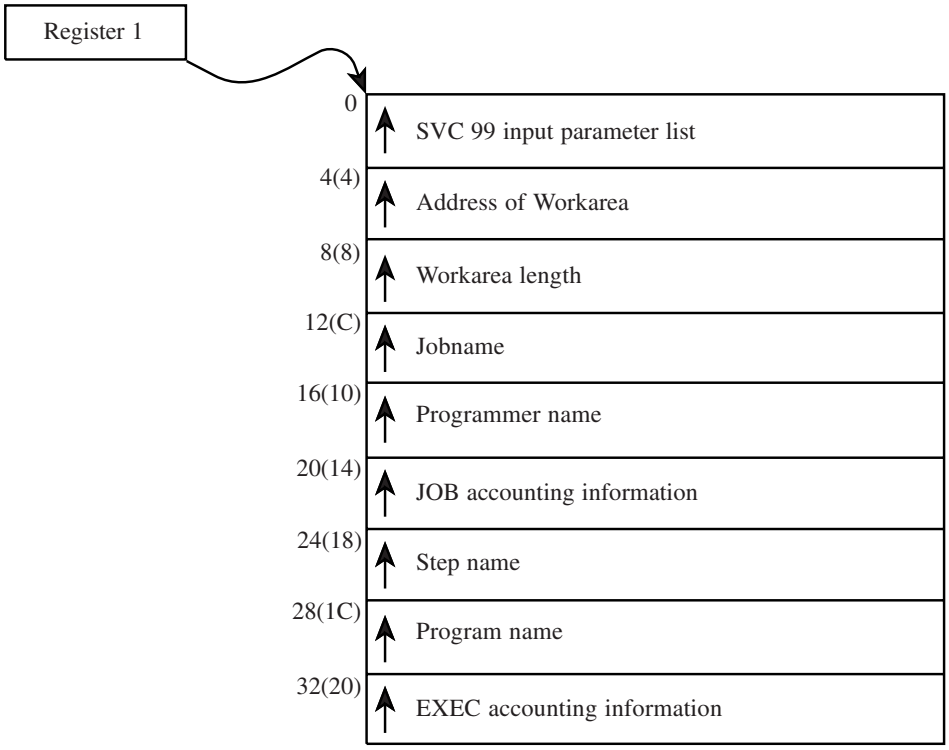

Figure 12. IEFDB401 Input Parameter Structure

Return Specifications

A return code from IEFDB401 indicates whether processing of the dynamic allocation request should continue.

If you associate multiple exit routines with IEFDB401, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-14 Restored to contents at entry

15 One of the following return codes:

Return Code
Explanation

0 Dynamic allocation request processing continues.

nonzero
Dynamic allocation request processing is terminated.

Coded Example of the Exit Routine

This sample exit routine first checks for an allocation verb code. If one is present, the routine loops through the text unit pointers to find the end of the text unit list. A return code is set to cancel the request if the unit description key is set to '3380'.

```
         TITLE 'DYNAMIC ALLOCATION INSTALLATION EXIT EXAMPLE'
IEFDB401 AMODE 31
IEFDB401 RMODE ANY
*****************************************************************
*                                                               *
*   $MOD(IEFDB401)                                              *
*                                                               *
*   DESCRIPTIVE NAME - DYNAMIC ALLOCATION                       *
*                         INSTALLATION EXIT EXAMPLE             *
*                                                               *
*   COPYRIGHT = 5665-291                                        *
*               THIS MODULE IS "RESTRICTED MATERIALS OF IBM"    *
*               (C) COPYRIGHT IBM CORP. 1987                    *
*               LICENSED MATERIALS - PROPERTY OF IBM.           *
*               REFER TO COPYRIGHT INSTRUCTIONS                 *
*               FORM NUMBER G120-2083                           *
*                                                               *
*   STATUS = HBB3310                                            *
*                                                               *
*   FUNCTION - THE DYNAMIC ALLOCATION FACILITY OF THE CONTROL   *
*              PROGRAM EXITS TO THIS 'EXIT ROUTINE' BEFORE      *
*              DOING ANY PROCESSING ON BEHALF OF A DYNAMIC      *
*              ALLOCATION REQUEST.  IT IS ENTERED FOR ALL       *
*              REQUESTS, FOREGROUND AND BACKGROUND.  THIS       *
*              ROUTINE MAY TEST AND MODIFY THE DYNAMIC          *
*              ALLOCATION INPUT, AND INDICATE THROUGH A         *
*              RETURN CODE WHETHER PROCESSING IS TO CONTINUE    *
*              OR IF THE REQUEST IS TO BE TERMINATED.           *
*                                                               *
*              THE ROUTINE MAY DELETE A TEXT UNIT BY ZEROING    *
*              THE TEXT UNIT POINTER OR TEXT UNIT KEY.          *
*              A WORK AREA IS PROVIDED TO FACILITATE THE        *
*              ADDITION OR MODIFICATION OF TEXT UNITS AND       *
*              IMMEDIATELY FOLLOWS THE TEXT UNIT POINTER        *
*              LIST.  POINTERS TO ITS ADDRESS AND LENGTH        *
*              ARE PASSED AS PARAMETERS.  THIS AREA IS IN       *
*              FETCH-PROTECTED SCHEDULER-KEY CORE.  IF TEXT     *
*              UNIT POINTERS ARE TO BE ADDED TO THE POINTER     *
*              LIST, THEY MUST BE ADDED TO THE END OF THE       *
*              LIST, (IN THE WORK AREA DESCRIBED ABOVE) AND     *
*              THE END-OF-LIST INDICATOR MUST BE ADJUSTED.      *
*                                                               *
*   DEPENDENCIES - THIS MODULE RECEIVES CONTROL IN SCHEDULER    *
*                  KEY FROM A PREVIOUS MODULE OF DYNAMIC        *
*                  ALLOCATION.                                  *
*                                                               *
*   MODULE TYPE - PROCEDURE                                     *
*                                                               *
*   PROCESSOR - ASSEMBLER H                                     *
*                                                               *
*   ATTRIBUTES - REFRESHABLE, SCHEDULER KEY, EXTENDED PLPA,     *
*                SUPERVISOR STATE, AMODE(31), RMODE(ANY)        *
*                                                               *
*   ENTRY POINT - IEFDB401                                      *
*                                                               *
*   LINKAGE - STANDARD ENTRY AND EXIT LINKAGE                   *
*                                                               *
```

```
*   INPUT - REGISTERS - REG0    -  IRRELEVANT                *
*                       REG1    -  PARAMETER LIST ADDRESS    *
*                       REG2-12 -  IRRELEVANT                *
*                       REG13   -  SAVE AREA ADDRESS         *
*                       REG14   -  RETURN ADDRESS            *
*                       REG15   -  ENTRY ADDRESS             *
*                                                            *
*         - DATA - UPON ENTRY REG1 POINTS TO A LIST OF       *
*                  ADDRESSES FOR THE FOLLOWING ITEMS:        *
*                                                            *
*                   1.  THE DYNAMIC ALLOCATION REQUEST BLOCK *
*                       IN SCHEDULER-KEY, FETCH-PROTECTED    *
*                       CORE.                                *
*                   2.  A POINTER TO THE WORK AREA THAT      *
*                       FOLLOWS THE TEXT UNIT POINTER LIST.  *
*                   3.  LENGTH OF THE WORK AREA              *
*                   4.  8-CHARACTER JOB NAME                 *
*                   5.  20-BYTE PROGRAMMER NAME              *
*                   6.  JOB ACCOUNTING INFORMATION - IN THE  *
*                       STANDARD FORMAT PASSED TO SMF EXITS. *
*                   7.  8-CHARACTER STEP NAME                *
*                   8.  8-CHARACTER PROGRAM NAME             *
*                   9.  STEP ACCOUNTING INFORMATION - IN THE *
*                       STANDARD FORMAT PASSED TO SMF EXITS. *
*                                                            *
*   EXITS - REGISTERS - REG0-13 - RESTORED                   *
*                       REG14   - RETURN ADDRESS             *
*                       REG15   - RETURN CODE                *
*                                                            *
*         - RETURN CODE - 0     - CONTINUE SVC 99 PROCESSING *
*                         non-0 - TERMINATE SVC 99 PROCESSING*
*                                                            *
*         - DATA - POSSIBLE CHANGES IN THE DYNAMIC ALLOCATION*
*                  REQUEST BLOCK, THE TEXT POINTERS, OR THE  *
*                  TEXT UNITS.                               *
*                                                            *
**************************************************************
         TITLE 'DYNAMIC ALLOCATION (SVC 99) PARM LIST'
         IEFZB4D0
         TITLE 'DYNAMIC ALLOCATION KEY TABLE'
         IEFZB4D2
         TITLE 'DYNAMIC ALLOCATION INSTALLATION EXIT EXAMPLE'
IEFDB401 CSECT
         SPACE 1
**************************************************************
*        STANDARD REGISTER EQUATES                           *
**************************************************************
         SPACE 1
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         SPACE 1
**************************************************************
*        STANDARD INPUT LINKAGE                              *
```

```
*************************************************************
         SPACE 1
         STM   R14,R12,12(R13)    SAVE CALLER'S REGISTERS
         BALR  R12,R0             ESTABLISH ADDRESSABILITY
         USING *,R12              TO CSECT
         EJECT
*************************************************************
*  INITIALIZE RETURN CODE, GET POINTER TO THE PARAMETER LIST  *
*  AND ESTABLISH ADDRESSABILITY TO THE SVC 99 REQUEST BLOCK.  *
*************************************************************
         SPACE 1
         XR    R15,R15            INITIALIZE RETURN CODE
         L     R9,0(R1)           ESTABLISH ADDRESSABILITY TO
         USING S99RB,R9           THE SVC 99 REQUEST BLOCK
         SPACE 1
*************************************************************
*  CHECK FOR AN 'ALLOCATION' VERB CODE, (X'01'), AND IF NOT   *
*  BRANCH TO CONTINUE THE REQUEST UNCHANGED.  ELSE ESTABLISH  *
*  ADDRESSABILITY TO THE TEXT UNIT POINTER LIST AND CHECK     *
*  FOR A VALID TEXT UNIT POINTER,non-0,AND THEN CHECK FOR THE *
*  END OF THE TEXT UNIT LIST.  IF NEITHER OF THE CHECKS PASS, *
*  THEN ADJUST THE TEXT UNIT POINTER AND LOOP UNTIL ONE DOES. *
*************************************************************
         SPACE 1
         CLI   S99VERB,S99VRBAL   IS THIS AN ALLOCATION VERB CODE?
         BNE   DB401D             BIN TO CONTINUE REQUEST UNCHANGED
         L     R8,S99TXTPP        ESTABLISH ADDRESSABILITY TO
         USING S99TUPL,R8         THE TEXT UNIT POINTER LIST
DB401A   DS    0H
         CLC   S99TUPTR,BLANKPTR  IS THIS A BLANK/¬LAST TEXT UNIT PTR?
         BE    DB401B             BIY TO CHECK THE NEXT TEXT UNIT PTR
         CLC   S99TUPTR,LSTBLKPT  IS THIS A BLANK/LAST TEXT UNIT PTR?
         BE    DB401D             BIY TO EXIT THE EXIT ROUTINE
         B     DB401C             GO CHECK THE TEXT UNIT CONTENTS
DB401B   DS    0H
         LA    R8,4(,R8)          POINT TO NEXT TEXT UNIT POINTER
         B     DB401A             GO CHECK THE NEXT TEXT UNIT POINTER
         SPACE 1
*************************************************************
*  ESTABLISH ADDRESSABILITY TO THE SVC 99 TEXT UNIT AND CHECK *
*  FOR A 'UNIT DESCRIPTION SPECIFICATION' KEY, (X'0015'), AND *
*  IF MATCHED CHECK FOR A '3380' PARAMETER.  IT THIS MATCHES, *
*  SET THE RETURN CODE TO CANCEL THE REQUEST, (X'04'), AND    *
*  RETURN TO CALLER.  IF EITHER OF THESE CHECKS FAIL THEN     *
*  BRANCH BACK TO CONTINUE THE LOOP OF TEXT UNIT POINTERS.    *
*************************************************************
         SPACE 1
DB401BB  DS    0H
         TM    S99TUPTR,LSTVALPT  IS THIS A VALID LAST TEXT UNIT PTR?
         BO    DB401D             BIY TO EXIT THE EXIT ROUTINE
         B     DB401B             GO GET NEXT TEXT UNIT POINTER
DB401C   DS    0H
         L     R7,S99TUPTR        ESTABLISH ADDRESSABILITY TO
         USING S99TUNIT,R7        THE SVC 99 TEXT UNIT
         CLC   S99TUKEY,UNDESKEY  IS THIS A UNIT DESCRIPTION KEY?
         BNE   DB401BB            BIN TO CHECK IF LAST TEXT UNIT PTR
         CLC   PARM3380,S99TUPAR  IS THIS A 3380 PARAMETER?
         BNE   DB401BB            BIN TO CHECK IF LAST TEXT UNIT PTR
         LA    R15,4(,R15)        SET RETURN CODE TO CANCEL REQUEST
DB401D   DS    0H
         EJECT
*************************************************************
*        STANDARD EXIT LINKAGE                              *
*************************************************************
         SPACE 1
         L     R14,12(R13)        RESTORE CALLER'S REGISTERS
```

```
         LM    R0,R12,20(R13)     EXCEPT REGISTER 15
         BR    R14                RETURN TO CALLER
         DROP  R7,R8,R9,R12
         SPACE 1
*****************************************************************
*        DECLARATIONS AND CONSTANTS                             *
*****************************************************************
         SPACE 1
PARM3380 DC    C'3380'            3380 PARAMETER
BLANKPTR DC    F'0'               CHECKS FOR A BLANK TEXT UNIT PTR
LSTBLKPT DC    X'80000000'
UNDESKEY DC    H'21'
LSTVALPT EQU   X'80'
         END
```

Chapter 26. IEFDOIXT — Edit / Check A Caller's Dynamic Output Text Units Exit

Topics for This Exit Appear as Follows:

Dynamic output is a system service that users can invoke by issuing the OUTADD or OUTDEL macro. Dynamic output allows an installation to specify the output characteristics of a sysout data set dynamically as an alternative to specifying these characteristics on the OUTPUT JCL statement. For more detailed information about using dynamic output, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

IEFDOIXT is the dynamic output installation exit. You can use IEFDOIXT to:

- Edit the text units. For example, you can use this exit to limit requests for COPIES to be less than or equal to 50.
- Detect requests for unsupported devices or features.
- Correct requests for unsupported devices or features by issuing installation-defined reason codes. These reason codes help the invoker of dynamic output diagnose problems that the exit detected.

Installing the Exit Routine

The IBM-supplied version of IEFDOIXT resides in SYS1.LINKLIB. You can replace IEFDOIXT in SYS1.LINKLIB with your own version or place your version of IEFDOIXT in an authorized library somewhere else in the search order prior to SYS1.LINKLIB. For more details on the search order for load modules, see *z/OS MVS Programming: Assembler Services Guide*.

For general instructions on installing an exit routine, see "Link editing an Installation Exit Routine into a Library" on page 3.

Exit Routine Environment

IEFDOIXT receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1 and must return control in the same state and key.

- In AMODE 31 and RMODE ANY.
- In the caller's address space.
- With no locks or resources held (other than storage obtained via GETMAIN and an ESTAE) when it calls IEFDOIXT.
- Under a type 3 SVC. See *z/OS MVS Programming: Authorized Assembler Services Guide* for a description of the restrictions in this environment.

Exit Recovery: IEFDOIXT runs under dynamic output's recovery environment. If recovery for dynamic output gets control during exit routine processing, dynamic output returns to its invoker and sets return and reason codes to indicate that the exit abnormally terminated.

Exit Routine Processing

Each time that you invoke dynamic output via the OUTADD or OUTDEL macro, it links to the IEFDOIXT installation exit. When IEFDOIXT completes its processing, it returns control to dynamic output. If no errors are detected, dynamic output then creates or deletes the output descriptor.

Dynamic output passes the exit a copy of the input from the caller of dynamic output. Before calling the exit, dynamic output performs the following verification of the caller's input data:

- Dynamic output references the input data in the key of the caller that issued the OUTADD or OUTDEL macro. If a protection exception (0C4 ABEND) occurs when the data is referenced, dynamic output passes control back to the caller, and the installation exit does not get control.
- Dynamic output verifies the text units. If the text units are not valid, dynamic output reports the problem as being that the parameter list is not valid. In general, dynamic output does not differentiate between text unit errors that the caller causes and those that the exit causes. In addition, because the exit works on a copy of the caller's parameter list, the caller can not recognize the changes that the exit has made to the text units. For this reason, the exit must ensure that text units are updated correctly.

If dynamic output determines that there is an error in the caller's input data, the installation exit does not receive control. If dynamic output verifies the caller's input data, the installation exit receives a copy of the data. This input data includes the fixed parameter list (DOCNP), the text units and a text unit pointer list with pointers to the copied text units. The exit also receives 500 bytes of storage so that it can update the caller's text unit pointers and text units.

In the exit routine, you can read or make valid alterations to the data passed to the exit. You can omit text units by zeroing their text unit pointers. You can add text units, as well as the text unit pointers for the new text units, in the work area provided. Concatenate the new text unit pointers to the end of the passed text unit pointer list, which is contiguous with the work area. If you add new text unit pointers, reset the high-order bit that indicates the last text unit pointer.

The exit should set the values of registers 0, 1 and 15 on return to dynamic output. The value of register 15 indicates whether dynamic output is to cancel or proceed with the request. Registers 1 and 0 are used for the exit to optionally pass diagnostic information to the dynamic output caller. For specification of the registers on return from the exit, see “Return Specifications” on page 173, and *z/OS MVS Programming: Assembler Services Guide*.

Upon return from the exit, dynamic output performs the following verification of the registers returned by the exit and the data areas passed to the exit:

- Dynamic output verifies the contents of registers 0, 1, and 15. If the register contents are not valid, or if the return code in register 15 is an 8, dynamic output denies the user's request and returns a reason code that identifies the problem.
- Dynamic output verifies the fixed parameter list (DOCNP) and reports any problems with the parameter list by setting the appropriate return and reason codes. Because dynamic output also verifies DOCNP before linking to the exit routine, unique error return and reason codes are issued to help differentiate between errors in DOCNP which the caller causes, and errors that the exit routine causes.
- Dynamic output verifies the text units. If the text units are not valid, dynamic output reports the problem as being that the parameter list is not valid. Dynamic output does not differentiate between text unit errors that the caller causes and those that the exit causes. In addition, because the exit works on a copy of the caller's parameter list, the caller can not recognize the changes that the exit has made to the text units. For this reason, the exit should make changes to the text units very carefully.

  When the exit alters the text units to contain negative values in the number of parameters field or in the length of parameter field, the results are unpredictable. Dynamic output only checks these fields before calling the exit routine. It is the exit's responsibility to make sure that it does not pass negative values in these fields. It also is essential that the exit does not cause an OUTADD request to contain no text units.

If no errors are detected, the output descriptor is created or deleted.

Programming Considerations

Code the IEFDOIXT routine to be reentrant.

The copy of the caller's input data, the text unit pointers and the 500-byte work area that are passed to the exit are in key 1 storage, subpool 229. This storage is pageable and fetch protected.

The IBM-supplied exit routine only zeroes the contents of registers 0, 1, and 15. It restores the other registers and returns control to dynamic output.

The exit is an authorized exit, so you must follow standard security and integrity procedures.

Dynamic output links to IEFDOIXT. The LINK macro does not restore the register contents, so you must be sure that you restore the contents of registers 2-13 before returning to dynamic output.

Tracing IEFDOIXT's Input and Output

You can use the generalized trace facility (GTF) to trace IEFDOIXT's input and output, when the GTF identifier for dynamic output is active. The GTF identifier for dynamic output is user event F62.

Each GTF trace record is prefixed with a 24-byte field that uniquely identifies the creator of the trace record and the trace record's sequence number. The format of the 24-byte field is as follows:

(Hex)	Contents
X'00'	'TCB ' (EBCDIC representation)
X'04'	TCB address of the task that invoked the SVC
X'08'	'SVRB' (EBCDIC representation)
X'0C'	SVRB address for this invocation of the SVC
X'10'	'SEQ#' (EBCDIC representation)
X'14'	Sequence number of trace record. That is, **1** indicates the **first** trace record of the input to IEFDOIXT, **2** indicates the **second** trace record, and so on.

Each GTF trace record can have a maximum length of 256 bytes. The following is a description of the GTF trace records for IEFDOIXT:

- The first trace record contains a character string header (starting at offset X'18') that indicates whether the data being traced is IEFDOIXT's input or output.
- The second trace record contains the following information:

(Hex)	Contents
X'18'	' DOCNP->'
X'20'	Pointer to copy of DOCNP
X'24'	' WORKAREA START->'
X'38'	Pointer to the start of IEFDOIXT's workarea
X'3C'	' WORKAREA END->'
X'4C'	Pointer to the end of IEFDOIXT's workarea

- The third record through the second-to-last record contains the data area that was passed to IEFDOIXT. This includes the copy of the SVC caller's parameters and the installation workarea. More than one trace record is issued if the data area exceeds the maximum length of one trace record (256 bytes).
- The last record contains a character string that indicates the end of the trace data.

See *z/OS MVS Diagnosis: Tools and Service Aids* for information on using GTF.

Entry Specifications

Dynamic output passes three address parameters to the installation exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of three consecutive fullwords. Each fullword is a pointer; the first fullword points to the parameter list (DOCNP), the second fullword points to the beginning (first byte) of the 500-byte work area that the exit can use, and the third fullword points to the end (last byte) of the work area. Figure 13 on page 173 shows the parameter structure.
2-12	Not applicable
13	Not applicable

14 Return address

15 Not applicable

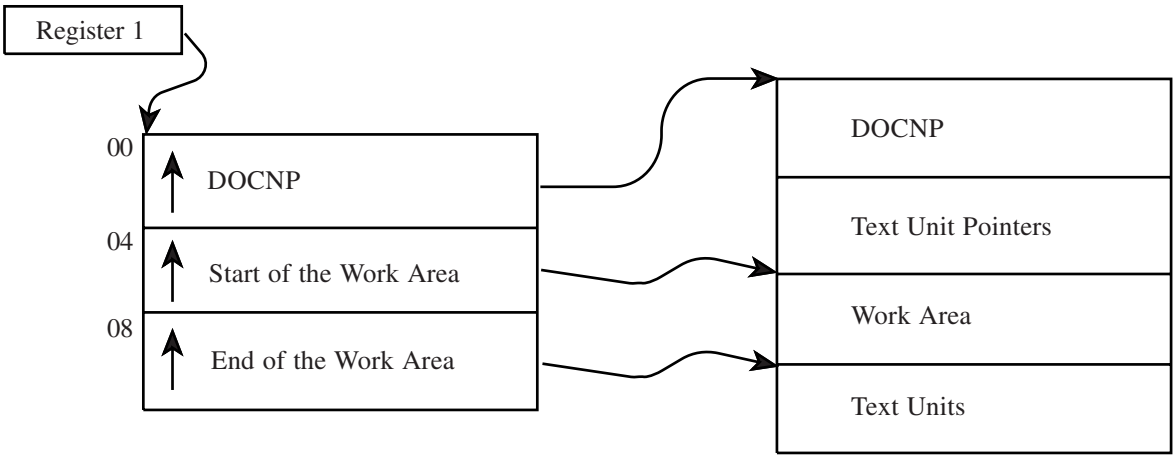

Figure 13. IEFDOIXT Input Parameter Structure

Parameter Descriptions: Register 1 contains the address of a pointer list that consists of three contiguous fullwords in storage. The first fullword points to the address of the fixed parameter list, the DOCNP, which is mapped by macro IEFDOCNP (data area DOCNP). For a mapping of the DOCNP data area, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

IEFDOIXT returns control to dynamic output and passes back a return code, a reason code and, optionally, a key in error.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0 The contents of register 0 depend on the return code that the exit routine places in register 15.
- If the return code is zero, register 0 must contain a zero.
- If the return code is 8, register 0 may contain a zero or an installation-defined reason code with a value between X'6000' and X'7FFF'.

1 The contents of register 1 depend on the return code that the exit places in register 15.
- If the return code is zero, set register 1 to zero.
- If the return code is 8, register 1 can contain either an erroneous text unit key in the two low-order bytes, or zero. The two high-order bytes must always contain zeroes.

2-13 Restored to contents at entry

14 Return address

15 Contains one of the following return codes:

Return Code	Description
0	The request is to be processed.
8	The request is to be denied. The values in register 0 and 1 are set accordingly and returned to the dynamic output caller in those registers.

Chapter 27. IEFJFRQ — Subsystem Function Request Exit

Topics for This Exit Appear as Follows:

You can use the IEFJFRQ exit to tailor the subsystem interface (SSI) processing of subsystem function requests. See *z/OS MVS Using the Subsystem Interface* for more information on subsystem function requests.

IEFJFRQ receives control at two different points in SSI function request processing:

- Prerequest — before the SSI routes either a directed or broadcast subsystem function request to the target subsystem(s). For broadcast requests, IEFJFRQ is called once for each subsystem defined to the SSI.
- Postrequest — after the SSI has routed either a directed or broadcast subsystem function request to all appropriate subsystems.

At the prerequest point, you can use IEFJFRQ to modify or suppress processing for any subsystem, or for any subsystem function request. (See *z/OS MVS Using the Subsystem Interface* for a list of subsystem function requests.) For example, you can:

- Suppress commands or WTOs.
- Prevent calls to a specific subsystem, for example, a subsystem that is failing repeatedly.
- Route directed requests to a different subsystem.
- Interrupt a broadcast request, that is, cause the SSI to discontinue routing the broadcast request to the subsystems that have not yet processed the request.
- Maintain records of subsystem invocation.
- Enforce installation policies for subsystem invocation.
- Modify the return code that the caller of the SSI receives.

At the postrequest point, you can use IEFJFRQ to:

- Update records to reflect the results of function requests acted on by multiple exit routines during the prerequest processing.
- Modify the return code that the caller of the SSI receives.

Applications using the IEFJFRQ installation exit can determine whether it is available by testing the JESFRQEX flag in the JESFLG field of the JESCT data area.

Controlling the Exit Routine through the Dynamic Exits Facility

IBM has defined the IEFJFRQ exit to the dynamic exits facility. You can refer to the exit by the name IEFJFRQ. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

You may install more than one exit routine at the IEFJFRQ exit point. Each exit routine can examine and modify the control blocks that represent the subsystem function request. The control blocks passed to the target subsystem reflect the modifications made by all installed exit routines.

You can control the order that exit routines receive control by using the FIRST or LAST parameter on one of the following services:

- The CSVYDYNEX REQUEST=ADD macro
- The SETPROG EXIT,ADD command
- The EXIT ADD statement of the PROGxx parmlib member.

You can use the ADDABENDNUM parameter on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. See *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* for more information on the CSVDYNEX macro, and *z/OS MVS System Commands* for more information on the SETPROG EXIT operator command. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine provides recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine. Note that IEFJFRQ cannot take advantage of consecutive abend processing, since IEFJFRQ supports fastpath calls in any PSW key,

You can use the DELETE parameter on the CSVDYNEX macro, or the SETPROG EXIT,DELETE command to delete exit routines from IEFJFRQ. If the delete request does not specify FORCE=YES, the system will not free the exit routine's storage, since the IEFJFRQ exit supports callers in any PSW key. See *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* for more information on the FORCE parameter on the CSVDYNEX macro, and *z/OS MVS System Commands* for more information on the FORCE parameter on the SETPROG command.

Exit Routine Environment

IEFJFRQ receives control in the following environment:

- Enabled for interrupts
- In the state and key of the caller of the SSI (issuer of the IEFSSREQ service that produced the subsystem function request that drives this exit)
- In AMODE 31
- With any locks held by the caller of the SSI

- In primary ASC mode
- In the dispatch mode (task or SRB) of the caller of the SSI
- In the cross-memory mode of the caller of the SSI.

Exit Recovery

You should consider the resources accessed by your exit routine and the impact on system performance, when evaluating whether your exit routine should provide its own recovery. IBM recommends that you establish your own recovery only if you access system resources, and only for subsystem function requests that concern your exit routine.

A system recovery routine will get control if:

- The exit routine abnormally ends and the exit routine does not provide its own recovery
- The error percolates beyond the exit routine's recovery routine

The system does not route the function request to any other exit routines associated with the exit, or to the target subsystem. If the error occurs while the SSI is processing a broadcast request, the SSI continues routing the request to any remaining subsystems. The abend processing of the dynamic exits facility determines whether the exit routine continues to be invoked.

Exit Routine Processing

When IEFJFRQ receives control, the SSI has:

- Validated the SSOB and SSIB control blocks to ensure that these control blocks are addressable and have the correct eyecatchers and lengths. Note that an SSOB may have more than one acceptable length.
- Set the SSOBSSIB field in the SSOB control block to point to the life-of-job SSIB, if the original subsystem function request did not specify an SSIB control block. See *z/OS MVS Using the Subsystem Interface* for more information on the life-of-job SSIB.

IEFJFRQ may be called multiple times for each subsystem function request. The process for calling IEFJRFQ differs depending on whether the request is directed or broadcast as follows:

- For directed requests, IEFJFRQ is called:
  - Once before the target subsystem receives control
  - Once after the target subsystem receives control

  Note that the postrequest instance is called even if the prerequest instance caused the SSI to bypass the call to the target subsystem.
- For broadcast requests, IEFJFRQ is called:
  - Once when the MSTR subsystem is invoked to initiate the broadcast processing
  - Once for each target subsystem
  - Once after broadcast processing is complete and the request has been routed to all appropriate subsystems.

  Note that the postrequest instance is called even if the prerequest instance caused the SSI to bypass all broadcast processing.

Prerequest processing:

IEFJFRQ receives control at the prerequest processing exit point before the SSI routes either a directed or broadcast subsystem function request to the target subsystem(s). Note that IEFJFRQ is invoked once for each subsystem receiving the request.

Postrequest processing:

IEFJFRQ receives control at the postrequest processing point after the SSI has routed either a directed or broadcast subsystem function request to all appropriate subsystems. Its primary purpose is to inform exit routines of the actions the SSI has taken with respect to the current subsystem function request.

Programming Considerations

IEFJFRQ exit routines must be reentrant.

Exit routines receive a pointer to the control blocks that represent the subsystem function request, which include the SSOB, SSIB, SSOB extension, and any areas pointed to from these control blocks. Exit routines should be careful when modifying any of these control blocks.

IEFJFRQ exit routines should not take actions that result in a call to the SSI, such as issuing a system command or dynamic allocation request. Actions that result in a call to the SSI could result in infinitely recursive calls to the IEFJFRQ exit. For example, an exit routine cannot issue an SVC WTO when processing a WTO/WTOR function request (SSI function code 9).

The SSI provides a 12-byte correlation token, named FRQP_CORRELATION_TOKEN, in the IEFJFRQP parameter mapping to assist exit routines in correlating calls resulting from a single subsystem function request. The correlation token has the following characteristics:

- The token is not valid if the system clock is not operating
- The first 8 bytes of the token are unique over the life of an IPL on a single system
- The 8-byte single system token concatenated with the 4-byte system ID in the parameter mapping form a 12-byte token that is unique across a sysplex.

Performance Considerations:

SSI processing, specifically the routing of subsystem function requests, may impact performance; therefore, consider the following recommendations so that system performance is not degraded:

- Exit routines installed at the IEFJFRQ exit point should not perform operations that may degrade system performance, such as; issuing WAIT requests, issuing requests for large amounts of dynamic storage, or issuing I/O requests. To reduce the need for storage requests, the system provides an area that your exit routine can use for dynamic storage. The FRQP_DYNSIZE constant in the IEFJFRQP parameter mapping defines the size of the dynamic storage area. The exit routine must clear the dynamic storage area before using it, because all exit routines installed at the IEFJFRQ exit point use the same work area.
- Exit routines should quickly determine whether the current subsystem function request is of interest, and return to the system immediately if not. Time consuming operations, such as obtaining storage, should be deferred until after this check. You can delay the establishment of recovery to perform this check, as

long as the check references only information identified by the input control blocks, that is, the IEFJFRQP parameter area, the SSOB, and the SSIB.

Entry Specifications

The IEFJFRQP macro maps the input to the IEFJFRQ exit, and contains the following information:

- The address of the SSOB control block representing the subsystem function request
- Flags that indicate:
  - Which instance (prerequest or postrequest) of the exit is being called
  - Whether the subsystem function request is a broadcast request
  - Whether the SSIB is a copy of the life-of-job SSIB or was provided by the SSI's caller.
- The current value of the return code that is passed back to the caller of the SSI, if IEFJFRQ does not change this value.
- A token that you can use to correlate exit calls resulting from a single subsystem function request.

For directed requests, the current return code value is always zero for the prerequest instance of the exit.

Every broadcast request begins as a subsystem function request directed to the MSTR subsystem. The SSI cannot determine if the request represents a broadcast request until the MSTR subsystem processes this request. Therefore, the broadcast indicator is not set when the SSI calls the prerequest instance of IEFJFRQ for a request directed to the MSTR subsystem, even if the request will eventually be a broadcast request.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	**Contents**
0	Does not contain any information for use by the exit routine
1	Address of a list of pointers
2-12	Does not contain any information for use by the exit routine
13	Register save area
14	Return address
15	Entry point address of IEFJFRQ exit routine

Parameter List Contents: Register 1 points to the following list of addresses, which are mapped by the FRQP_PLIST_AREA field of the IEFJFRQP macro:

- The address of the IEFJFRQP parameter area
- The address of the working storage that the system provides for use by the exit routine.

Note: The high-order bit is set in the address of the last parameter to indicate the end of the parameter list.

Return Specifications

The exit routine must specify the following return codes:

- A return code passed in register 15 that controls the system's processing of the subsystem function request
- A return code in register 0 that is passed to the caller of the SSI.

For the prerequest instance of the IEFJFRQ exit, the return code passed in register 0 for the caller of the SSI is used only if the return code passed in register 15 indicates that the SSI should not pass the subsystem function request to the target subsystem.

For the postrequest instance of IEFJFRQ, the return code passed in register 0 is passed back to the caller of the SSI. You can preserve the return code that would be returned by the SSI by copying FRQP CURRENT SSI RETCODE in register 0. Register 0 must contain one of the return code values defined in the IEFSSOBH mapping macro.

If you associate multiple exit routines with IEFJFRQ, you can specify how the return information is handled by using the ATTRIB KEEPRC function on one of the following:

- The SETPROG EXIT command
- The EXIT statement of the PROGxx parmlib member
- The CSVDYNEX macro services.

If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

Note that the KEEPRC function specifies tests to be performed on the return code set in register 15 on exit from the IEFJFRQ exit routine. The results of the tests control the information that is returned to the SSI, that is, both the exit's return code in register 15 and the SSI caller's return code in register 0.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine with the largest return code value. If multiple exit routines return with the same value, the value in the exit routine that finished first will be returned.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0	Return code to be provided to the caller of the SSI
1	The exit routine does not have to place any information in this register, and does not have to restore its contents to what they were when the exit routine received control.
2-14	Restored to contents on entry
15	One of the following return codes:

Return Code	Explanation
0	Route the subsystem function request to the target subsystem

4 Do not route the subsystem function request to the target subsystem

8 Do not route the subsystem function request to the target subsystem, or for broadcast requests, do not route the subsystem function request to any remaining subsystem that has not yet processed the request.

24 Do not route the subsystem function request to the target subsystem, or for broadcast requests, do not route the subsystem function request to any remaining subsystem that has not yet processed the request. Do not call any remaining IEFJFRQ exit routines.

The IEFJFRQP macro provides mnemonic names for the return code values.

Coded Example of the Exit Routine

A sample IEFJFRQ exit routine is provided in SYS1.SAMPLIB (in member IEFJSXIT).

Chapter 28. IEFUAV — User Account Validation Exit

Topics for This Exit Appear as Follows:

You can use the IEFUAV installation exit routine to validate the accounting information of users of APPC/MVS transaction programs (TPs).

IEFUAV is invoked:

- When the IBM-supplied APPC/MVS transaction scheduler (ASCH) selects a TP for execution and
- At the points in TP processing where APPC/MVS could have tailored the accounting information for the TP user. For example, IEFUAV is invoked when APPC/MVS tailors the account number to which resources are charged for that instance of the TP.
- The Workload Manager (WLM) component of the base control program is used to create forked/spawned address spaces. In addition to APPC/MVS TPs, the IEFUAV exit now receives control for forked/spawned address spaces.

When the IEFUAV exit receives control for a forked/spawned address space, the TP flag value indicator is set to 1. The jobname in the ASCB is BPXAS, which is the name of the procedure used to start the MVS initiator associated with forked/spawned address spaces.

Based on whether the TP user's accounting information is valid, IEFUAV sets a return code to indicate that either:

- Processing should continue for the unit of work or
- The unit of work is to be cancelled.

IEFUAV also allows you to place a message into a user's APPC/MVS job log. With the message you can provide information to supplement the return code.

For APPC/MVS TPs, IEFUAV is the only exit that allows you to validate accounting information for specific users **at execution time**. Therefore, it is recommended that you use IEFUAV, instead of IEFUJV (or another exit, such as

IEFUJI or IEFUSI), to validate the accounting information of APPC/MVS TPs. Even though IEFUJV is invoked (as part of C/I text processing) during the reading of a TP profile, the user of the TP is not known at that time.

Defining the Exit in SMFPRMxx

To allow the system to invoke IEFUAV, define the exit in the SMF parmlib member (SMFPRMxx). Specify IEFUAV on the EXITS option of the SUBSYS parameter for the ASCH subsystem. If your installation chooses not to define a SUBSYS parameter for ASCH, you can specify IEFUAV on the EXITS option of the SYS parameter.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFUAV installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFUAV or SYSyyy.IEFUAV. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

To define IEFUAV to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for IEFUAV, you need only define this exit in SMFPRMxx.

If you do not associate any exit routines with exit IEFUAV in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFUAV).

If you associate exit routines with IEFUAV in PROGxx, the system does not use the default exit routine. If you need this exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFUAV receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1.
- In AMODE 31 and runs RMODE ANY.
- In the address space of the unit of work being started

- With no locks or ENQs held.

Exit Recovery

IBM strongly recommends that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of IEFUAV.

If you do not provide recovery for IEFUAV, or if the exit routine's error percolates beyond its recovery routine, a system recovery routine will get control.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

The system invokes the IEFUAV exit routine during the processing of an APPC/MVS TP at the points in which the accounting information might have been altered. At these points, an installation can validate the user's accounting information before processing the transaction further. For example, IEFUAV is invoked when a TP issues Get_Transaction because this service allows APPC to tailor the accounting information for the unit of work, such as the account number to which a resource is charged.

When IEFUAV Is Invoked: IEFUAV will get control only for APPC/MVS TPs whose profiles specify TAILOR_ACCOUNT(YES). IEFUAV is invoked once for standard TPs and multiple times for multi-trans TPs, as follows:

- For both standard and multi-trans TPs, IEFUAV is invoked when the APPC/MVS transaction scheduler (ASCH) selects a TP for execution. In this case, IEFUAV receives the generic accounting information that is specified in the TP user's RACF profile.
- For multi-trans TPs only, IEFUAV is invoked when the TPs issue calls to the following services:
  - Get_Transaction, to obtain the next transaction. IEFUAV receives accounting information derived from the TP user's RACF profile.
  - Return_Transaction, to restore the generic “shell” environment that was established when the TP was initialized. IEFUAV receives generic accounting information from the TP user's RACF profile.

For information on creating a TP profile, see *z/OS MVS Planning: APPC/MVS Management*.

The circumstances under which IEFUAV receives control might impose restrictions on exit routine processing; see macros and restrictions in “Programming Considerations” on page 186 for more details.

Using IEFUAV to Validate Account Numbers

Code your IEFUAV routine to check the user's account number against a list of valid installation account numbers.

Using the Exit Function Code: IEFUAV determines why it was invoked by checking the exit function code in Word 3 of the parameter list (pointed to by Register 1). An installation would probably want IEFUAV to perform validation when Word 3 is set to a value of either 1 (generic account number) or 2 (possible account number alteration).

For exit function code 3, IEFUAV can simply return to the caller, as this code indicates the generic account number that IEFUAV already validated during TP initialization (exit function code 1).

See parameter descriptions in "Entry Specifications" on page 187 for more information on the exit function codes.

Validating the User's Account Number: IEFUAV receives, as parameters, the accounting information specified on the JOB statement in the TP user's RACF profile. The accounting information appears in a formatted list, and follows the order in which the accounting parameters were specified on the JOB statement. Table 4 on page 157 shows the format of the accounting information.

After locating the account number field, IEFUAV would typically do these checks:

- Is the account number present? A value of 0 in the first byte of the account number field indicates the account number was omitted.
- Does the account number have the correct length? A nonzero value in the first byte indicates the length of the account number.
- Is the account number valid? IEFUAV can determine this by comparing the account number to a list of valid account numbers.

At the end of its processing, IEFUAV sets a return code that indicates whether processing for this user should continue or be cancelled.

Placing a Message into a User's APPC/MVS Job Log: IEFUAV allows you to place a message into a user's APPC/MVS job log. Through the message, you can provide information to supplement the return code from IEFUAV. On entry, word 5 of the input parameter list points to a two-word area representing a message area. Place the length of the message in the first word. If the length exceeds zero, then the system issues the message that the second word points to regardless of the value returned in register 15. The maximum allowed length is 80.

The second word is the address of an 80-byte buffer; in that buffer, place the message to be issued to the APPC/MVS job log.

Programming Considerations

SMF provides a replaceable module for an unused exit.

If an installation includes an IEFUAV exit routine, the following programming standards must be observed:

- IEFUAV must follow standard linkage conventions.
- IEFUAV must be reentrant.
- IEFUAV cannot access ISAM data sets.

Macro Instructions and Restrictions are as follows.

- If you want to issue a WTOR macro from IEFUAV, also issue the WAIT macro with LONG=YES.
- You can issue any macros from IEFUAV, including the OPEN and DYNALLOC macros. The circumstances under which IEFUAV receives control, however, determine the results of OPEN and DYNALLOC processing:
  - When IEFUAV gets control because the APPC initiator started a standard or multi-trans TP, IEFUAV can open only those data sets specified in the APPC initiator's procedure. If IEFUAV needs access to other data sets, it may

dynamically allocate them, which adds those data sets to the APPC initiator's environment. Those data sets will remain available to the APPC initiator unless IEFUAV unallocates them before completing its processing.

– When IEFUAV gets control because a multi-trans TP issued either Get_Transaction or Return_Transaction, IEFUAV can open only those data sets that are specified in the TP's profile. If IEFUAV needs access to other data sets, it may dynamically allocate them, which adds those data sets to the TP's environment. Those data sets will remain available to the TP unless IEFUAV unallocates them before completing its processing.

Entry Specifications

IEFUAV is passed a list of parameter addresses (pointed to in register 1).

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point of IEFUAV

Parameter Descriptions: Register 1 points to a list of addresses. The list is specified in VL format (that is, the last address has its leftmost bit set to 1). The following describes the list of parameters:

Word 1
Address of the common exit parameter area. For details, see "Common Exit Parameter Area" on page 155.

Word 2
Address of a 20-byte area containing the programmer's name (in EBCDIC). This area is aligned left and padded with blanks if necessary.

Word 3
Address of a 1-byte area that contains a value (binary). The value indicates one of the following conditions:

Exit Function Code	Condition
0	This code is reserved, and no action is taken.
1	For both standard and multi-trans TPs, this code indicates that IEFUAV has received control during the initialization of a TP. The exit receives generic, installation-supplied accounting information from the TP user's RACF profile (in the area pointed to by Word 4).
2	For multi-trans TPs, this code indicates that accounting information might have been altered. The exit receives accounting information from the TP user's RACF profile (in the area pointed to by Word 4).

3 For multi-trans TPs, this code indicates a return to processing on behalf of the multi-trans "shell". The exit receives generic, installation-supplied accounting information from the TP user's RACF profile (in the area pointed to by Word 4).

4-255 These codes are reserved, and no action is taken.

Word 4

Address of an area containing accounting information. (See "Accounting Information" on page 157.)

Word 5

Address of a two-word area representing a message area. The first word contains a message length of zero on entry. The second word contains the address of an 80-byte message buffer. In that buffer, the exit routine can place a message to be issued to the APPC/MVS job log.

Note: The high-order bit is set in the address of the last parameter to indicate the end of the parameter list.

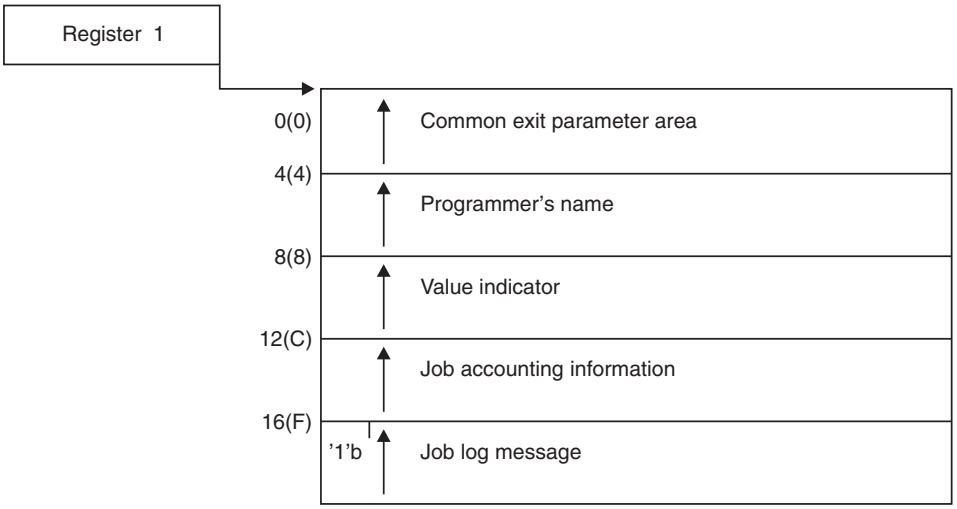

Figure 14. IEFUAV Input Parameter Structure

Return Specifications

A return code from IEFUAV indicates whether processing is to continue.

If you associate multiple exit routines with IEFUAV, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFUAV, and any of those exit routines return with a value of 8, the system will cancel the unit of work.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0 Not applicable

1 Address of input parameter list

Words 1-4
Not applicable

Word 5
Points to a two-word area representing a message to be written to the APPC/MVS job log. The first word contains the length of the message. The second word is the address of an 80-byte buffer; the buffer contains the text of the message.

2-14 Not applicable

15 One of the following return codes:

Return Code
Explanation

0 Processing is to continue.

8 This unit of work should be cancelled.

Coded Example of the Exit Routine

IBM provides a sample IEFUAV exit routine in SYS1.SAMPLIB (member SMFEXITS). The sample routine validates the account number of the current user.

If this routine meets the needs of your installation, use it instead of coding your own routine (you will need to make minor modifications to the sample). If you plan to code your own routine, you might want to refer to this routine as an example.

Chapter 29. IEFUJI — Job Initiation Exit

Topics for This Exit Appear as Follows:

IEFUJI receives control before the system selects a job on the input queue for initiation. A return code from IEFUJI indicates whether job processing is to continue or be cancelled.

You can use IEFUJI to:

- Validate job accounting information.
- Determine how long a job was on the input job queue before it was selected.
- Write particular information to an installation data set.

Defining the Exit in SMFPRMxx

In the SMF parmlib member (SMFPRMxx), specify IEFUJI on the EXITS option of either the SYS or SUBSYS parameters, depending on the scope of work (system-wide or subsystem-wide) the exit is to affect.

If you use the SUBSYS option, the system invokes the IEFUJI routine only for work running under the subsystems you specify on SUBSYS. If you use the SYS option, the system invokes the IEFUJI routine for work running under any SMF-defined subsystem, such as JES2, JES3, STC, ASCH, OMVS, or TSO.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFUJI installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFUJI or SYSyyy.IEFUJI. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the

EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its routines.

To define IEFUJI to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for this exit, you need only define IEFUJI in SMFPRMxx.

If you do not associate any exit routines with exit IEFUJI in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFUJI).

If you associate exit routines with IEFUJI in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFUJI receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31.
- In the address space of the job that is being started.
- With no locks or ENQs held.

Exit Recovery

IBM strongly recommends that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of IEFUJI.

An ESTAE-type recovery routine is set up by the module that calls IEFUJI; the recovery routine, if it gets control, will allow the job to continue processing if the exit routine abnormally ends.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

IEFUJI receives control from the system whenever a job on the input queue is selected for initiation. If the system availability manager (SAM) function is active and operational, the SAM job/step initialization exit routine executes before IEFUJI is called.

IEFUJI has information from the JOB statement available as parameters. The accounting information is in a formatted list, so that, for account number processing, IEFUJI is easier to use than exit IEFUJV. Table 4 on page 157 shows the format of the JOB statement accounting information.

At the end of its processing, IEFUJI sets a return code to indicate whether job processing is to continue or not.

Programming Considerations

SMF automatically provides a replaceable module for IEFUJI. If an installation includes IEFUJI, it must follow certain programming standards:

- The exit routine must follow standard linkage conventions.
- Code the exit routine reenterable and refreshable.
- IEFUJI can perform dynamic allocations and write to installation-defined data sets. In foreground jobs, data sets are allocated dynamically. However, for background jobs, you can either allocate data sets dynamically or you can pre-define (pre-allocate) a data set with a DD statement in the initiator cataloged procedure.
- IEFUJI cannot access ISAM data sets.
- Do not use a WTO with a routing code of 11 to send a message to the JESYSMSG data set for started tasks or TSO users.
- Do not use subpool 240 or 250 when obtaining storage for this exit. Using these subpools may result in errors when the exit receives control for address spaces that are created with the KEEPRGN attribute.
- To provide a consistent environment for accessing and allocating data sets across calls to SMF exits for the duration of a job or task, IEFUJI receives control with the initiator's JSCB active.

At job or job step termination, use the termination indicators in SMF record types 4, 5, 30, 34, and 35 to indicate that IEFUJI cancelled the job.

Macro Instructions and Restrictions: When issuing a WTOR macro, specify LONG=YES on the WAIT macro. Do not use a WTO with a routing code of 11 to send a message to the JESYSMSG data set for started tasks or TSO users.

Entry Specifications

The system provides a list of parameter addresses that IEFUJI can use.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEFUJI

Parameter Descriptions: Register 1 points to the following list of addresses:

Word 1

Address of the common exit parameter area. For details on the parameter area, see Table 3 on page 156.

Word 2

Address of a 20-byte area containing the programmer's name (in EBCDIC) from the JOB statement. This area is aligned left and padded with blanks if necessary.

Word 3

Address of a one-byte area indicating (in binary) the requested job selection priority. The value of this field equals the user-assigned priority of 0 to 14 (taken from the PRTY parameter on the JOB statement).

Word 4

Address of an area containing the accounting information from the JOB statement. (See "Accounting Information" on page 157.)

Word 5

Address of a 4-character area that contains the name of the subsystem for the job being processed. Examples:

- ASCH, JES2, or JES3 — indicates the name of the subsystem that selected the job
- OMVS — indicates a forked or spawned address space
- STC — indicates a started task
- TSO — indicates a time sharing option task
- The jobname — used if it is four or fewer characters and none of the above apply

Note: The high-order bit is set in the address of the last parameter to indicate the end of the parameter list.

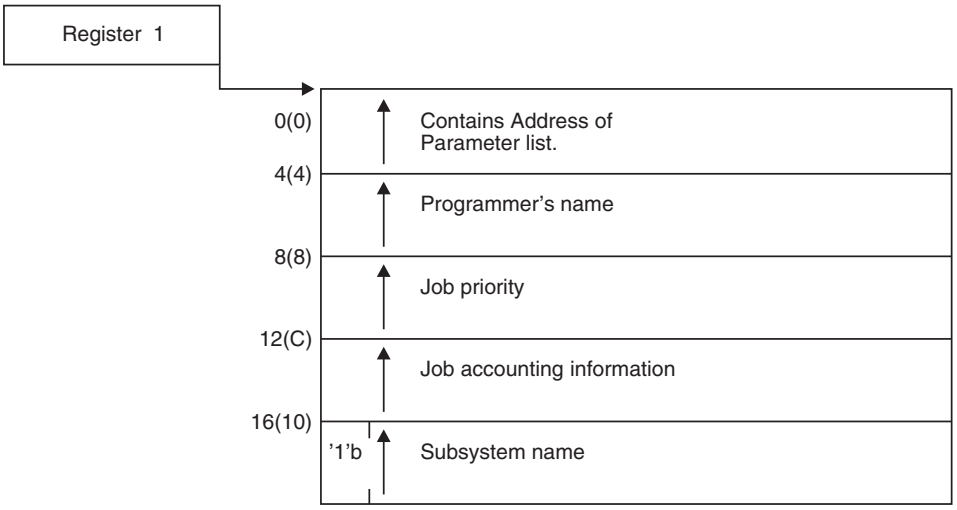

Figure 15. IEFUJI Input Parameter Structure

Return Specifications

A return code from IEFUJI indicates whether job processing is to continue.

If you associate multiple exit routines with IEFUJI, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG

EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFUJI, and any of those exit routines return with a value of 4, job processing will not continue.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-14 Not applicable

15 One of the following return codes:

Return Code
Explanation

Value of 4
Job processing is to be cancelled.

Value other than 4
Job processing is to continue.

Coded Example of the Exit Routine

A sample IEFUJI exit routine is provided in SYS1.SAMPLIB (in member SMFEXITS). This routine determines how long a job has been on the input job queue before it is initiated. It then writes this value and the job priority to the SMF data set as a user record.

Chapter 30. IEFUJP — Job Purge Exit

Topics for This Exit Appear as Follows:

IEFUJP receives control from the job entry subsystem (JES2 or JES3) when a job is ready to be purged from the system, meaning that the job terminated and the system has written all SYSOUT output pertaining to the job. A return code from IEFUJP indicates whether the SMF job purge record (type 26) is to be written to the SMF data set.

Defining the Exit in SMFPRMxx

To allow the system to invoke IEFUJP, define the exit in the SMF parmlib member (SMFPRMxx). Specify IEFUJP on the EXITS option of the SUBSYS parameter for the STC subsystem. If your installation chooses not to define a SUBSYS parameter for STC, you can specify IEFUJP on the EXITS option of the SYS parameter.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFUJP installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFUJP or SYSyyy.IEFUJP. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

To define IEFUJP to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for this exit, you need only define IEFUJP in SMFPRMxx.

If you do not associate any exit routines with exit IEFUJP in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFUJP).

If you associate exit routines with this exit in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFUJP receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1.
- In AMODE 31.

Exit Recovery

IBM strongly recommends that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of IEFUJP.

An ESTAE-type recovery routine is set up by the module that calls IEFUJP; the recovery routine, if it gets control, will allow the job to continue processing if the exit routine abnormally ends.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

IEFUJP receives control from the job entry subsystem SMF writer. The addresses of the SMF common exit parameter list and the type 26 SMF record (SMF job purge record) are parameters that the exit routine can use to determine whether or not to write the SMF record.

At the end of its processing, IEFUJP sets a return code to indicate to the JES SMF writer whether to write the SMF record.

Programming Considerations

SMF automatically provides a replaceable module for an unused exit. If an installation includes IEFUJP, certain programming conventions must be followed:

- The exit routine must follow standard linkage conventions.
- Code the exit routine to be reentrant and refreshable.
- If you use installation-defined data sets with IEFUJP, you must define them with a DD statement in the job entry subsystem cataloged procedure.

Macro Instructions and Restrictions: When issuing a WTOR macro, specify LONG=YES on the WAIT macro. Do not use a WTO with a routing code of 11 to send a message to the JESYSMSG data set for started tasks or TSO users.

Entry Specifications

The job entry subsystem provides the addresses of the SMF record and the common exit parameter area for the exit routine to use.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Contains address of parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEFUJP

Parameter Descriptions: Register 1 points to the following list of addresses:

Word 1
The address of the common exit parameter area. (See Table 3 on page 156.)

Word 2
The address of an area containing the SMF job purge record (type 26) to be written to the SMF data set.

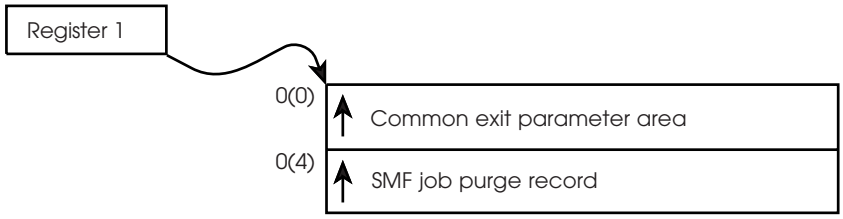

Figure 16. IEFUJP Input Parameter Structure

Return Specifications

A return code from IEFUJP indicates whether the SMF type 26 record is to be written.

If you associate multiple exit routines with IEFUJP, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple

exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFUJP, and any of those exit routines return with a value of 4, the system will not write the SMF type 60 record.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-14 Restored to contents at entry

15 One of the following return codes:

Return Code
Explanation

Value of 4
Job purge record is not to be written to the SMF data set.

Value other than 4
Job purge record is to be written to the SMF data set.

Chapter 31. IEFUJV — Job Validation Exit

Topics for This Exit Appear as Follows:

IEFUJV receives control at three different points in the converter/interpreter (C/I) processing of an input stream. They are:

1. Preconversion — before each job control statement (or cataloged procedure) in the input stream is converted.
2. Postconversion — after all job control statements for a job have been converted.
3. Postinterpretation — after all job control statements for a job have been interpreted.

For z/OS UNIX System Services, this exit is called only once when each BPXAS initiator is started. It is not called for forked/spawned requests and can not be used to validate them.

A return code from this exit indicates whether job processing is to continue.

At the preconversion point, you might use IEFUJV to:

- Validate any accounting fields included in the JOB and EXEC statements (except symbolic parameters) by comparing them to a standard list.
- Validate or assign the REGION request.
- Validate or assign job TIME and job step TIME parameters.
- Control output stream data by using the OUTLIM or SPACE parameters.
- Check for authorization to use restricted data sets.
- Create user-written records.
- Assign the user identification to be included in both the SMF job/step termination record and the SMF job purge record.
- Limit the size of temporary data sets handled by VIO.
- Require checkpoint/restart for jobs requesting a large amount of processor time.
- Enforce installation standards on usage of the ADDRSPC parameter.

- Override certain JES initialization parameters (such as designation of where SWA blocks are to be obtained) that are passed to converter routines.

At the postconversion point, you might use IEFUJV to:

- Create user-written records.
- Assign the user identification to be included in both the SMF job/step termination record and the SMF job purge record.
- Override certain JES initialization parameters (such as designation of where SWA blocks are to be obtained) that are passed to converter routines.

At the postinterpretation point, you might use IEFUJV to:

- Create user-written records.
- Assign the user identification to be included in both the SMF job/step termination record and the SMF job purge record.

Defining the Exit in SMFPRMxx

In the SMF parmlib member (SMFPRMxx), specify IEFUJV on the EXITS option of either the SYS or SUBSYS parameters, depending on the scope of work (system-wide or subsystem-wide) the exit is to affect.

If you use the SUBSYS option, the system invokes the IEFUJV routine only for work running under the subsystems you specify on SUBSYS. If you use the SYS option, the system invokes the IEFUJV routine for work running under any SMF-defined subsystem, such as JES2, JES3, STC, ASCH, OMVS, or TSO.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFUJV installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFUJV or SYSyyy.IEFUJV. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines. However, you cannot use the JOBNAME parameter of the SETPROG EXIT command to restrict exit IEFUJV processing to a particular job.

To define IEFUJV to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for this exit, you need only define IEFUJV in SMFPRMxx.

If you do not associate any exit routines with exit IEFUJV in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFUJV).

If you associate exit routines with this exit in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the

SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFUJV receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0 or 1 (based on the caller's key). When the entry code (contained in word 3 of the input parameter list) is 32, the key is 0.
- In AMODE 31.

In a JES2 environment, conversion might take place on one processor and interpretation of the same job on another. Therefore, the IEFUJV exits could receive control on different processors for the same job. In that case, timing comparisons of the job flow would not be valid.

For an interpreter call in a JES2 environment, a security environment must be established if the exit is to obtain access to any protected resources. For example, if a command is to be issued with RACF OPERCMDS active, pass a user security token (UTOKEN) on the MGCRE macro to establish authority for the user requesting access to the command. See *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU* for information about the MGCRE macro.

Exit Recovery

IBM strongly recommends that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of IEFUJV.

An ESTAE-type recovery routine is set up by the module that calls IEFUJV; the recovery routine, if it gets control, will allow the job to continue processing if the exit routine abnormally ends.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

There are two data formats related to JCL statements. The first data format is the JCL "card image." A JCL card image is an 80-character EBCDIC string that represents either an entire JCL statement or a portion of a JCL statement that is continued. The second data format is the C/I text string representation of the JCL statement. The C/I text format consists of an established pattern of hexadecimal codes, or keys, assigned to each parameter or subparameter. See Part 4, “MVS Converter / Interpreter Text Processing,” on page 357 for more details about C/I text strings and the appropriate processing exit points.

JES provides several exit points related to JCL card image processing. See *z/OS JES2 Installation Exits* and *z/OS JES3 Customization* for additional information about this processing.

Note that IEFUJV receives control following the JES exits related to card image processing. Your installation is responsible for coordinating the processing between IEFUJV and the JES exits.

Note further that if you use the IEFUJV exit to change certain parameters on the JOB statement, the result might be that the internal representation of the JCL card image would reflect the changes while the job itself continues to be processed according to the original JOB statement. For example, if you change the CLASS parameter as part of the exit routine, the job will still run in its original specified class. This applies to both the JES2 and JES3 environments. These other JOB statement parameters have the same restriction: GROUP, MSGCLASS, NOTIFY, PASSWORD, PRTY, SECLABEL, TYPRUN, and USER.

Preconversion processing: IEFUJV receives control at the preconversion processing exit point before each JCL statement card image is converted. The input parameter list provides an indication of the type of JCL statement being processed.

The exit will be invoked multiple times for a continued JCL statement (once per card image). The JCL statement type indicator is the same for each card image of the continued JCL statement.

When modifying a JCL statement, the updated JCL statement must adhere to the JCL syntax as defined in *z/OS MVS JCL Reference*. The following updates are not permitted at the preconversion processing point:

- Do not include additional JCL statements
- Do not add continuation card images.
- Do not change the operation field on a JCL statement.
- Do not change the identifier field on a JCL statement.

If a procedure is used, it is expanded before the IEFUJV exit routine receives control. For example, for a cataloged procedure, the sequence of statements are:

```
//UJV     JOB
//STEP1   EXEC  PROC=MYPROC
XXMYPROC  PROC
XXSTEP2   EXEC  PGM=...
```

followed by the other statements of the procedure. Note that the resolved values for symbolic parameters are not passed to the IEFUJV exit routine.

Using IEFUJV for Job Accounting: You might want to use an IEFUJV exit routine for job accounting. If so, consider the following:

- For APPC/MVS transaction programs (TPs), IBM recommends that you use IEFUAV instead of IEFUJV to validate accounting information. IEFUAV is the only exit that allows you to validate the accounting information of a TP user **at execution time**. Even though IEFUJV is invoked when a TP profile is created (specifically, when the profile's JCL statements are processed by the converter/interpreter), the TP user is not known at that time. Therefore, when you need to validate the TP user's accounting information (such as the job name or account number), use the IEFUAV exit routine. See Chapter 28, “IEFUAV — User Account Validation Exit,” on page 183 for more information.

- Depending upon the processing to be performed, it may be more efficient to check JOB and EXEC statement accounting fields in the IEFUJI exit routine and the first IEFUSI exit routine, respectively. The accounting fields are passed as parameters to IEFUJI and IEFUSI, making a statement scan routine unnecessary. Either of these exit routines can assign user identification, and the IEFACTRT exit routine can write messages to JOBLOG.
- When running JES2, you can use Exit 03, the job statement accounting field scan exit (HASPRSCN), as well as the IEFUJV exit. Because Exit 03 receives control before the IEFUJV exit, do not use the IEFUJV exit to change the following fields of the JOB statement: CLASS, MSGCLASS, NOTIFY, PRTY, PASSWORD and TYPRUN.
- If an installation checks JOB statement accounting in the IEFUJV exit for all tasks, the IEFUJV exit should not be taken, unless modified, for started tasks. Started tasks do not have any JOB statement accounting and might be cancelled by the installation exit.

For jobs cancelled by IEFUJV from the converter, only SMF record types 6 and 26 are generated.

Preconversion input limitations: The following information is not made available to this exit routine:

- Resolved values for JCL symbolic parameters
- JCL COMMAND, Command, and Comment statements
- JES2 control statements
- JES3 control statements

Postconversion processing: IEFUJV receives control at the postconversion processing point once per job to indicate that the conversion processing for the job has completed.

Postinterpretation processing: IEFUJV receives control at the postinterpretation processing point once per job to indicate that the interpretation processing for the job has completed.

When running JES3, a JES3 user can use JES3 installation exits, in addition to the IEFUJV exit, to write programs to examine and change the results of interpreter processing and allow the job to proceed or to flush the job from the system. For more information about the JES3 installation exits, see *z/OS JES3 Customization*.

Programming Considerations

When issuing a WTOR macro, specify LONG=YES on the WAIT macro. Do not use a WTO with a routing code of 11 to send a message to the JESYSMSG data set for started tasks or TSO users.

IEFUJV must be reenterable and refreshable because PLPA pages are stolen. That is, they can be paged in but not paged out, and subsequent page-ins overlay any code changes.

To use installation-defined data sets with this exit routine, you must define them with a DD statement in the job entry subsystem cataloged procedure. When running JES2, you must also define the data sets with a DD statement in the initiator cataloged procedure.

IEFUJV cannot access ISAM data sets.

Exit IEFUJV is called multiple times for a job. When you change the exit routine after it has been called at least once, but before all the calls for the job have been made, the job might be rejected because of incorrect JCL. When you rerun the job it will run successfully.

Entry Specifications

The converter/interpreter passes to IEFUJV a list of parameter addresses (in register 1).

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEFUJV

Parameter Descriptions: Register one points to the following list of addresses:

Word 1
The address of the common exit parameter area. (See Table 3 on page 156.)

Word 2
When the value pointed to by Word 3 is neither 16 nor 32, Word 2 is the address of the JCL statement card image.

This word is 0 when the value pointed to by word 3 is 16 or 32.

Word 3
The address of a 1-byte area that indicates the specific exit processing point. For preconversion, it also indicates the type of JCL statement being passed to the exit. See "Exit Routine Processing" on page 203 for more information. The indicator is one of the following binary values:

Value	Meaning
0	indicates the preconversion exit point; null statement card image.
1	indicates the preconversion exit point; JOB statement card image.
2	indicates the preconversion exit point; EXEC statement card image.
4	indicates the preconversion exit point; DD statement card image.
8	indicates the preconversion exit point; PROC statement card image from cataloged procedure.
16	indicates the postconversion exit point; all JCL has been converted.
32	indicates the postinterpretation exit point; all JCL has been interpreted.
64	indicates the preconversion exit point; JCL definition table defined (JDT) statement card image.

128 indicates the preconversion exit point; extended JCL statement type card image. Extended JCL statement type refers to any new JCL statements defined as of MVS/ESA Version 4. (For possible exceptions see "Exit Routine Processing" on page 203.)

Word 4

This word is 0 when the value pointed to by word 3 is 32.

When the value pointed to by word 3 is not 32, word 4 is the address of the JES initialization parameters that are passed to the converter routine. The address points to the first converter parameter field, which is a 1-byte bit-map defined as follows for bits that are set on:

.......1 Programmer name required.

......1. Account number required.

.....1.. Indicates that a job is enabled to run with the SWA located in virtual storage above 16 megabytes.

Note: IEFUJV may turn any of these bits on or off at any time except when the value pointed to by word 3 is 32. If a bit is turned on or off more than one time, the final setting of the bit is the one which will be honored.

Note: The account number must not be required in the exit for started task JOB statements because there is no way to put an account number on a started task JOB statement. An accounting number can be required on an EXEC statement in SYS1.PROCLIB.

Word 5

The address of a 4-character area that contains the name of the subsystem for the job being processed. Examples:

- ASCH, JES2, JES3 or OMVS - indicates the name of the subsystem that selected the job (For OMVS, see Guideline below.)
- STC - indicates a started task
- TSO - indicates a time sharing option task
- The jobname - used if it is four or fewer characters and none of the above apply

Guideline: The first time after an IPL that a z/OS UNIX fork or spawn occurs, z/OS UNIX creates JCL for a job named BPXYOEJS using a default JOB statement and passes that JCL to the MVS Converter and Interpreter (C/I). For this one job, C/I calls IEFUJV under SUBSYS = OMVS. Job BPXYOEJS does not actually execute. Instead the Interpreter creates SWA blocks for this job and passes them back to z/OS UNIX, which stores them for later use. These SWA blocks will be used by all subsequent forked or spawned address spaces. This means that whatever version of IEFUJV is active when the first fork or spawn occurs will be the only IEFUJV entered for forked or spawned address spaces, even if a new version of IEFUJV is later dynamically activated (through the SETPROG EXIT command). The BPXAS initiators in which the forked or spawned processes run, do go through IEFUJV, but with SUBSYS = STC exit.

Word 6

The address of a 4-byte area that contains the environment indicator associated with the subsystem specified in word 5.

The value that applies to all subsystems is 0.

Value Meaning

0	Default - “no meaning”

The values that apply to ASCH (APPC Scheduler) are 1, 2, and 3.

Value	Meaning
1	APPC Scheduler Utility TP Add call
2	APPC Scheduler Utility TP Retrieve call
3	APPC Scheduler Utility TP Reconvert call

Note: The high-order bit is set in the address of the last parameter to indicate the end of the parameter list.

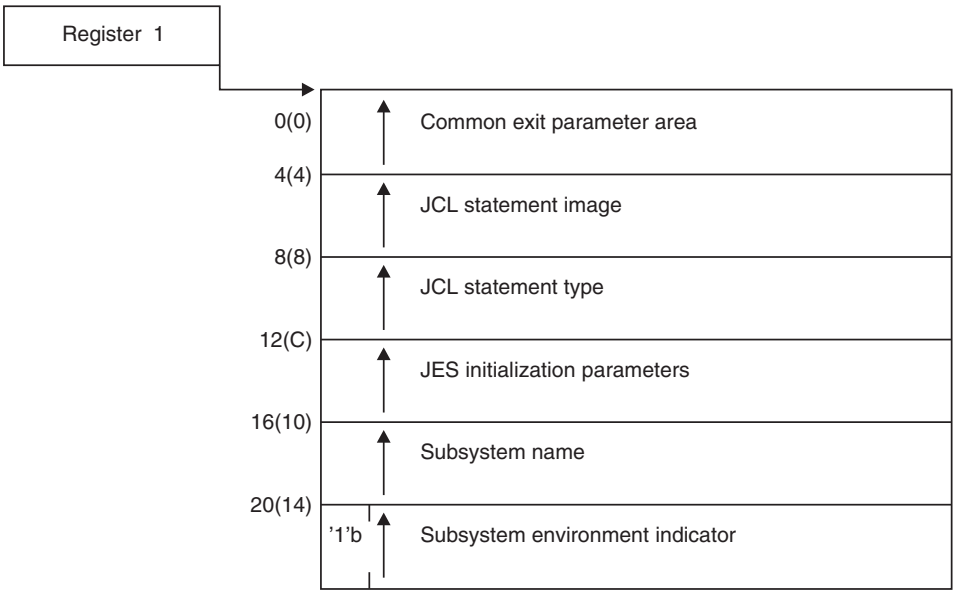

Figure 17. IEFUJV Input Parameter Structure

Return Specifications

A return code from IEFUJV indicates whether job processing will continue or be terminated.

If you associate multiple exit routines with IEFUJV, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFUJV, and any of those exit routines return with a value of 4, job processing will be terminated.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-14 Restored to contents at entry

15 One of the following return codes:

Return Code
Explanation

Value of 0
Job processing is to continue.

Value of 4
Job processing is to be cancelled.

Coded Example of the Exit Routine

Sample IEFUJV exit routines are provided in SYS1.SAMPLIB in members SMFEXITS and IEEUJV. The routine in SMFEXITS checks the validity of a continued JOB statement and of values supplied for the REGION, PRTY, TIME, and accounting parameters in the JOB statement. The routine uses characters from the account number to index a table that contains allowable values for these parameters. If any value is not valid, the sample IEFUJV routine terminates the job.

The sample in IEEUJV changes the SYSOUT class to SYSOUT=* for jobs in specified JOB classes and for specified SYSOUT classes. Assembled into the exit routine is a list of eligible job classes and a list of eligible SYSOUT classes. Thus, if a job enters the system in one of the specified job classes and contains a DD statement specifying SYSOUT=class, where class is one of the specified SYSOUT classes, then the SYSOUT class will be changed to '*'.

Chapter 32. IEFUSI — Step Initiation Exit

Topics for This Exit Appear as Follows:

IEFUSI receives control before each job step is started (prior to allocation). A return code from this exit indicates whether the job step is to be started or the job should be cancelled.

You can use IEFUSI to:

- Validate job step accounting information.
- Write to a user data set.
- For long-running jobs, create and write a user step-initiation SMF record in case of system failure.
- Set the region size and region limit for all programs that run under this job step. For more information about controlling region size and region limit, see *z/OS MVS Initialization and Tuning Guide*.

  Note: For programs with the NOHONORIEFUSIREGION Program Property Table (PPT) attribute specified, region and MEMLIMIT values and limits set by the IEFUSI exit are not honored. See the *z/OS MVS Initialization and Tuning Reference* and the *z/OS MVS Initialization and Tuning Guide* for more information about this PPT attribute and how it affects the IEFUSI-altered region and MEMLIIT settings.
- Set limits on the use of data spaces and hiperspaces created by application programs with storage key 8-F.
- Limit the number of pages that can be shared at one time through the use of the IARVSERV macro.
- Set the default size of data spaces and hiperspaces.
- Limit the use of the 16 exabyte address space above two gigabytes.

- Reduce the values of LDAELIM, LDAEVVRG, LDALIMIT, and LDAVVRG. Initially, these fields contain the maximum amount of storage available to the user. Specifically:

```
   LDAELIM = LDAEVVRG = 32Mb
and
   LDALIMIT = LDAVVRG = LDASIZA - 64K
```

  Note: For programs with the NOHONORIEFUSIREGION Program Property Table (PPT) attribute specified, these region values are not changed by the IEFUSI exit. See the *z/OS MVS Initialization and Tuning Reference* and the *z/OS MVS Initialization and Tuning Guide* for more information about this PPT setting and how it affects the IEFUSI-altered region and MEMLIMIT values and limits.

If an installation uses major and minor account numbers with several fields, IEFUSI is easier to use than IEFUJV for account number processing because the accounting fields are placed in a formatted list. See Table 4 on page 157 for the format of the accounting information.

Limiting Region Size: There are several factors to consider in using IEFUSI to limit region size:

- If IEFUSI is not available, or IEFUSI is not used to set up the region size, the system will use IEALIMIT.
- To use IEFUSI for region size control, you must tell the system to bypass the IEALIMIT exit by setting a flag in the IEFUSI parameter list.
- When writing the installation exit, the region size should be made less than the region limit. This is to protect against programs that issue variable requests for storage with very large upper bounds and then do not immediately free part of that space, or free such a small amount that a subsequent request for storage (possibly issued by a system service) causes the job to fail. See *z/OS MVS Initialization and Tuning Guide* for a discussion on the relationship between region size and region limit and how the system uses these values.
- For programs with the NOHONORIEFUSIREGION Program Property Table (PPT) attribute specified, IEFUSI-altered region and MEMLIMIT values and limits are not honored. The NOHONORIEFUSIREGION PPT attribute can be specified in the SCHEDxx member of SYS1.PARMLIB, or as an IBM supplied PPT default. This PPT attribute is used to bypass IEFUSI region controls for programs that require more region space to successfully execute. See the *z/OS MVS Initialization and Tuning Reference* and the *z/OS MVS Initialization and Tuning Guide* for more information on this PPT setting and how it affects the IEFUSI-altered region and MEMLIMIT values and limits.

Although you can use the IEFUSI exit to modify the region size of an address space, IBM strongly recommends that you do not alter the region size of address spaces in the OMVS subsystem category.

Note: A default IEFUSI module is included in SYS1.LPALIB. This default module does not do any processing, but just returns to the caller. It will be picked up first by the system at IPL time, unless you code your exit specifically in the PROGxx parmlib member using the EXIT ADD statement.

Comparing IEFUSI with IEALIMIT when Limiting Region Size

Historically, users could limit program storage below 16 megabytes in virtual storage by using IEALIMIT. IEALIMIT can still be used to limit program storage in the nonextended region; however, IEFUSI is the preferred exit routine, and has the following advantages over IEALIMIT:

- IEFUSI is a separate load module in the link pack area. You must supply a routine named IEFUSI and linkedit it into LPALIB or an LPALSTxx member of SYS1.PARMLIB. IEALIMIT must reside in the nucleus, so you must linkedit the nucleus every time you replace IEALIMIT with a new version. You must linkedit your routine again into the nucleus each time you IPL a different version of the nucleus, as all versions of the nucleus initially contain the IBM-supplied IEALIMIT routine.
- IEFUSI users can obtain information required to set a region size and region limit. IEALIMIT scans system control blocks to gather that information; thus IEFUSI is easier to write and less susceptible to system changes.
- IEALIMIT requires that the local lock be held and therefore cannot issue SVCs. IEFUSI has neither of these restrictions.
- IEFUSI can control the region size and region limit of both the area above and the area below 16 megabytes in virtual storage. IEALIMIT can set values for only the area below 16 megabytes, leaving values for the extended private area above 16 megabytes to be determined by the system.
- Previously, if the region requested by a job was greater than 16 megabytes, MVS allowed a minimum of 32 megabytes in the extended region; IEFUSI can override this 32-megabyte minimum value.

Defining the Exit in SMFPRMxx

In the SMF parmlib member (SMFPRMxx), specify IEFUSI on the EXITS option of either the SYS or SUBSYS parameters, depending on the scope of work (system-wide or subsystem-wide) the exit is to affect.

If you use the SUBSYS option, the system invokes the IEFUSI routine only for work running under the subsystems you specify on SUBSYS. If you use the SYS option, the system invokes the IEFUSI routine for work running under any SMF-defined subsystem, such as JES2, JES3, STC, ASCH, OMVS, or TSO.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFUSI installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFUSI or SYSyyy.IEFUSI. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its routines.

To define IEFUSI to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for IEFUSI, you need only define this exit in SMFPRMxx.

If you do not associate any exit routines with exit IEFUSI in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFUSI).

If you associate exit routines with IEFUSI in PROGxx, the system does not use the default exit routine. If you need the default exit routine, add it explicitly to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFUSI receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31.

Exit Recovery

IBM strongly recommends that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of IEFUSI.

An ESTAE-type recovery routine is set up by the module that calls IEFUSI; the recovery routine, if it gets control, will allow the job to continue processing if the exit routine abnormally ends.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

IEFUSI exit routines receive control before the initiator checks to determine whether the job step will actually execute. That is, IEFUSI receives control before the system performs testing for COND and IF/THEN/ELSE/ENDIF, including ABENDs. This means that IEFUSI will always be called, regardless of whether the step will execute.

Using IEFUSI to Limit Data Space and Hiperspace™ Use: Your IEFUSI exit routine can communicate the values you want to use to limit data space and hiperspace use for programs with storage key 8-F. Code your routine to place these values in word 7 of the SMF parameter list before returning control. (See the parameter descriptions in “Entry Specifications” on page 215 for a description of word 7.)

Your IEFUSI exit routine can include region processing for the private area, both less than and greater than 16 megabytes. The exit checks the region requested on the JOB or EXEC JCL statement, and determines whether it is acceptable. You can

then communicate the values you want to use to limit access to private area storage. Your IEFUSI routine places these values in the appropriate fullwords in the SMF parameter list before returning control to the initiator.

For more information on data spaces and hiperspaces, see *z/OS MVS Programming: Extended Addressability Guide*.

Programming Considerations

When coding an IEFUSI exit routine, observe the following conventions:

- When issuing a WTOR macro, specify LONG=YES on the WAIT macro. Do not use a WTO with a routing code of 11 to send a message to the JESYSMSG data set for started tasks or TSO users.
- Do not use subpool 240 or 250 when obtaining storage for this exit. Using these subpools may result in errors when the exit receives control for address spaces that are created with the KEEPRGN attribute.
- To provide a consistent environment for accessing and allocating data sets across calls to SMF exits for the duration of a job or task, IEFUSI receives control with the initiator's JSCB active.
- IEFUSI must be reenterable and refreshable because PLPA pages are stolen. That is, they can be paged in but not paged out, and subsequent page-ins will overlay any code changes.
- IEFUSI can perform dynamic allocations and write to installation-defined data sets. In foreground jobs, data sets are allocated dynamically. However, for background jobs, you can either allocate data sets dynamically or you can pre-define (pre-allocate) a data set with a DD statement in the initiator-cataloged procedure.
- IEFUSI cannot access ISAM data sets.

Additional Considerations for z/OS UNIX Applications: When running z/OS UNIX applications you need to consider that fork and spawn are issued to create new address spaces. The default processing on fork and spawn is for the z/OS UNIX kernel to propagate the region size from the parent to the child. Because the region size in the parent process has already passed through IEFUSI and has an approved region size, IBM recommends that you bypass normal IEFUSI processing when the subsystem (Word 8) is OMVS.

At the time of IEFUSI processing, the kernel has not yet propagated the parent's region size to the child, so IEFUSI has nothing to work with. If IEFUSI modifies the region size of the child process, the kernel will honor that region size and not propagate the region size from parent to child. This can result in failure of a fork if the region size is insufficient in the child to capture the parent's storage.

Entry Specifications

The system passes a list of parameter addresses to IEFUSI.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	**Contents**
0	Not applicable
1	Address of the parameter list

2-12 Not applicable

13 Register save area

14 Return address

15 Entry point address of IEFUSI

Parameter Descriptions: Register one points to the following list of addresses:

Word 1
The address of the common exit parameter area. (See Table 3 on page 156.)

Word 2
The address of an eight-byte area containing the job step name (in EBCDIC) from the EXEC statement. This area is aligned left and padded with blanks if necessary.

For a forked/spawned address space, this will be STEP1 when the address space is first started and will be *OMVSEX, after an exec().

Note: It is possible that some address spaces will be associated with IEESYSAS. For that case, JMRJOB will contain IEESYSAS.

Word 3
The address of an eight-byte area containing the program name (in EBCDIC) from the EXEC statement. This area is aligned left and padded with blanks if necessary. If you refer back, the area contains **pgm=*.DD**. For a forked/spawned address space,this will be BPXPRFC; after an exec(), it will be BPXPRECP.

Word 4
The address of an area containing the accounting information from the EXEC statement. (See "Accounting Information" on page 157.)

Word 5
The address of a six-word area that IEFUSI can use to communicate with MVS regarding the region size and region limit it desires. The region size and region limit information consists of:

Sub-word 1
Flag word to specify that IEFUSI, rather than IEALIMIT, is to provide the information on how to control access to private area storage.

The flag word is initialized to zero. Your routine sets the flag bits to mean the following:

Bit	Value	Meaning
0	0	IEALIMIT is supplying information on how to control access to private area storage.
	1	IEFUSI is supplying information on how to control access to private area storage.
1	0	Check if the requested below-16-megabyte-region size is available. If the size is not available an 822 abend occurs.
	1	Do not check if the requested below-16-megabyte-region size is available. If the size is not available, a less-predictable abend than 822 might occur.
2	0	Do not check if the requested above-16-megabyte-region size is available. If the size is not available, a less-predictable abend than 822 might occur.
	1	Check if the requested above-16-megabyte-region size is available. If the size is not available, an 822 abend occurs.
3-31		Reserved

Note: The settings for bits 1 and 2 have opposite meanings. You must turn on bit 2 when the amount of contiguous free space requested is critical for the step to be executed.

Sub-word 2
Region size request on the JOB or EXEC JCL statement. For a forked address space, this shows as 54M.

Sub-word 3
Region limit below 16 megabytes.

Sub-word 4
Region size below 16 megabytes.

Sub-word 5
Region limit above 16 megabytes.

Sub-word 6
Region size above 16 megabytes.

On every entry to IEFUSI, the last four words in the region size information list are set to X'FFFFFFFF'.

There is no lower bound on the region limit and region size that IEFUSI can request. If the JCL specifies REGION=0 with no MEMLIMIT coded, and the IEFUSI exit changes the REGION size but does not set the MEMLIMIT, the MEMLIMIT value is set to the REGION size above 16MB.

Word 6
The address of a word containing a flag indicating a V=R job.

Word 7
The address of a four-word area containing IBM-supplied default values for data spaces, hiperspaces, and data sharing (through the IARVSERV macro). These defaults apply only for programs running in problem state with user keys. The number of sharing pages, address spaces, data spaces and hiperspaces can be limited only for jobs running in user keys and in problem state. Jobs running in system key (0-7) or in supervisor state may use unlimited sharing pages, data spaces and hiperspaces. The words are defined as follows:

Sub-word	Contents
1	Default data space and hiperspace size. It is specified in blocks of 4K bytes and must be in the range of 1-X'00080000'. The IBM-supplied default is 956K (X'000000EF' x 4K).
2	Maximum combined size for all user key data spaces and hiperspaces that are created by application programs owned within an address space (in megabytes). The IBM-supplied default and the maximum that can be specified is (2**24)-1 megabytes.
3	Maximum number of user-key data spaces and hiperspaces created by application programs that can exist at any given time for an address space. The IBM-supplied default and the maximum number that can be specified is (2**32)-1.
4	Maximum number of source and target shared pages that can be used at one time by problem state callers using the IARVSERV SHARE services. The IBM-supplied default is 32 (which allows at

most 16 pages to be shared with 16 other pages; the number of pages include source plus target). The maximum you can specify is (2**31)-1.

Note: If you change the default, make sure you balance this with other uses of SQA storage. The number of 4K pages that Extended SQA (ESQA) requires equals (the number of shared pages plus the number of shared views plus 252) all divided by 127 and then ignoring any remainder.

Word 8

The address of a 4-character area that contains the name of the subsystem for the job being processed. Examples:

- ASCH, JES2, or JES3 — indicates the name of the subsystem that selected the job.
- OMVS — indicates a forked or spawned address space
- STC — indicates a started task
- TSO — indicates a time sharing option task
- The jobname — used if it is four or fewer characters and none of the above apply

Word 9

The address of an area consisting of three 64-bit fields used to specify MVS MEMLIMIT value. The MEMLIMIT information consists of:

Sub-parm 1

A 64-bit flag word. The first 8 bits indicate whether the source of the MEMLIMIT is from JCL, or the SMF-supplied system default. The remaining 56 bits are not used.

Your routine can set the flag bits to the following possible values:

Hex value of first 8 bits	Meaning
01 ('00000001'b)	MEMLIMIT is from SMF
02 ('00000010'b)	MEMLIMIT is from JCL
03 ('00000011'b)	MEMLIMIT was set to NOLIMIT because JCL specified REGION=0
FF ('11111111'b)	MEMLIMIT is from SMF (indicative of internal processing errors)

Note: When you initialize a child address space in the UNIX System Services environment, the source MEMLIMIT value can be set using definitions other than those provided through the IEFUSI exit. For additional information, see

- Handling process limits in *z/OS UNIX System Services Planning*.
- In *z/OS UNIX System Services Programming: Assembler Callable Services Reference*:
  - setrlimit (BPX1SRL).
  - spawn (BPX1SPN, BPX4SPN).

Sub-parm 2

The 64-bit MEMLIMIT originally requested by the source that is specified in the flagword. This value is specified in megabytes.

Sub-parm 3

The 64-bit MEMLIMIT requested by the IEFUSI exit. The initial value is X'FFFFFFFFFFFFFFFF' to indicate that no value was set by the exit. This value is specified in megabytes.

Note:

1. A MEMLIMIT of NOLIMIT is equivalent to X'00000FFFFFFF000'.
2. Critical address spaces are exempt from the IEFUSI imposed limit on above the bar virtual memory, so as not to affect system availability.

For a complete description of MEMLIMIT, and the ways to define it, see *z/OS MVS Programming: Extended Addressability Guide*.

Note: The high-order bit is set in the address of the last parameter to indicate the end of the parameter list.

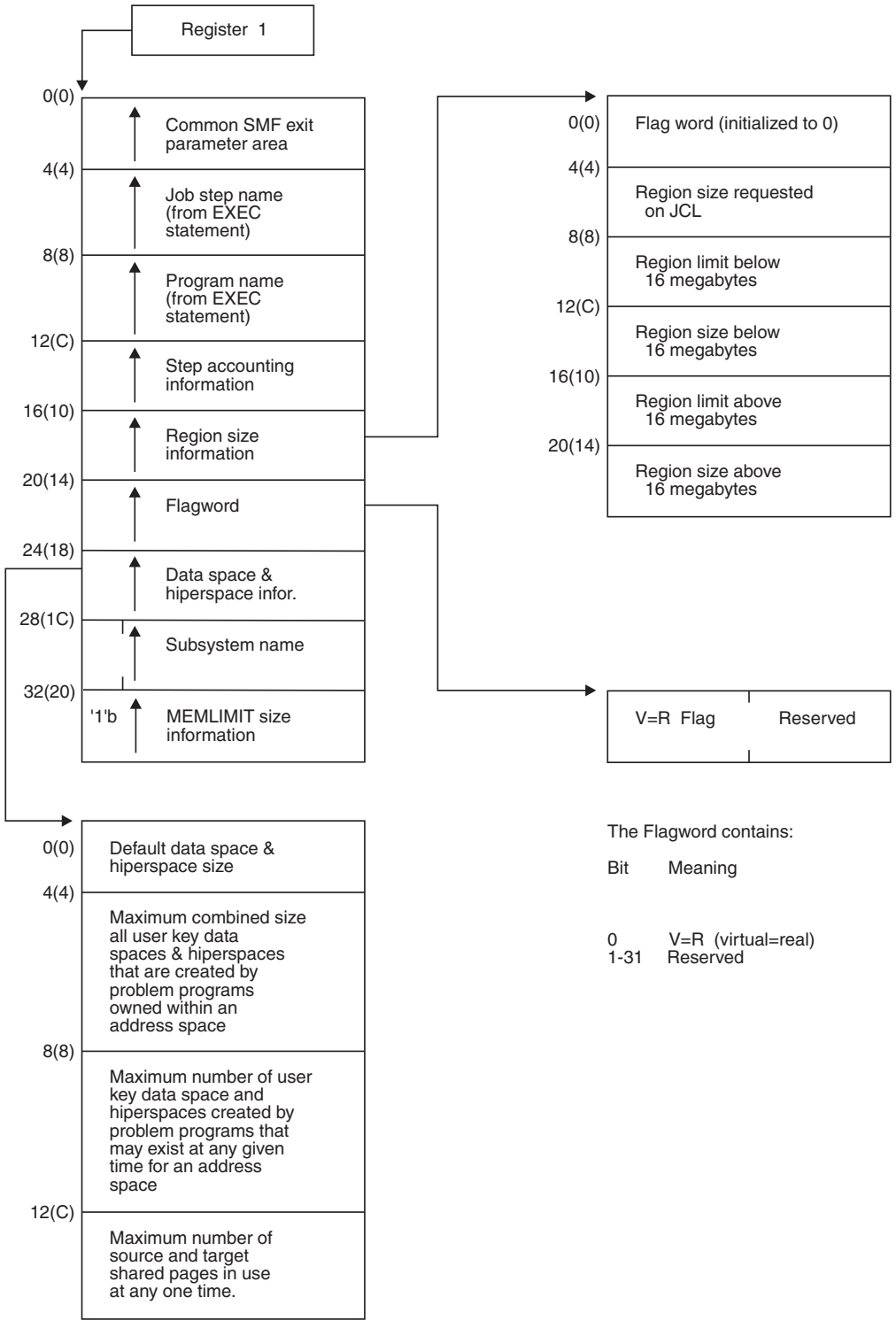

Figure 18. IEFUSI Input Parameter Structure

Return Specifications

A return code from IEFUSI indicates whether job processing should continue or be cancelled.

If you associate multiple exit routines with IEFUSI, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFUSI, and any of those exit routines return with a value of 4, job processing will be cancelled.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-14 Restored to contents at entry

15 One of the following return codes:

Return Code
Explanation

4 Job processing should be cancelled.

Oher than 4
Job processing should continue.

Examples

The examples that follow illustrate actual allocations based on region size and limit values set by IEFUSI.

Example of using IEFUSI to Limit Region Size: Suppose you want to limit all jobs in a given step-accounting category to a user region of 4 megabytes below and 4 megabytes above 16 megabytes. You also want to set a GETMAIN limit of 6 megabytes below 16 megabytes and 48 megabytes above 16 megabytes.

The system applies the following limits when allocating space for the program whose values you set in IEFUSI:

```
(Assume that the user's private area below 16 megabytes is 8
megabytes, and that the extended private area, above 16
megabytes, is approximately 1975 megabytes.)
```

Limit Value below 16Mb = 6Mb
(Less than 8Mb)

Limit Value above 16Mb = 48Mb
(The value from IEFUSI is greater than 32Mb, but less than the extended private area)

Region Size below 16Mb = 4Mb
(Less than limit value)

Region Size above 16Mb = 4Mb
(Less than extended limit value)

Examples of Storage Allocations Based on Values Set by IEFUSI: Assume that application program A has the following characteristics:

```
Limit value                150K
REGION size value          100K
Space currently allocated   80K
```

Program A issues the following variable length GETMAIN requests, in the order indicated (note that the GETMAIN requests are **cumulative**):

1. Request 5K—10K: 10K is allocated, making the currently allocated space 90K.

   Because the amount still unallocated (20K, relative to the region size of 100K), was greater than the maximum amount requested, the maximum amount was allocated.
2. Request 5K—100K: 10K is allocated, making the currently allocated space 100K.

   Because the amount still unallocated (10K, relative to the region size) was between the minimum and maximum requested, the unallocated space was allocated.
3. Request 40K—100K: 40K is allocated, making currently allocated space 140K.

   Although the amount still unallocated (0K, relative to the region size) was less than the minimum amount requested (40K), the minimum amount requested would not increase the currently allocated space beyond the limit value, so the minimum amount was allocated.
4. Request 15K—50K: the GETMAIN fails.

   The amount still unallocated (0K, relative to the region size) was less than the minimum amount requested (15K), AND the minimum requested would increase the currently-allocated space to 155K, which exceeds the GETMAIN limit value of 150K.

The region size value is usually set up to be less than the limit value. This will protect against programs that issue variable length GETMAINs with very large maximums and then do not immediately free part of that space, or free such a small amount that a subsequent GETMAIN (possibly issued by a system service) causes the job to fail.

As an example, suppose that the region size value equals the limit value, and a program issues a variable length GETMAIN with a maximum of 2 gigabytes - 1. If the GETMAIN is satisfied, all the space in the region up to the limit value will be allocated, and any subsequent GETMAIN that cannot be satisfied from free space in an already-existing subpool will cause the job to fail.

If, however, the region size value is less than the limit value, only space up to the region size value is allocated for the GETMAIN. Thus, an amount of space equal to the limit value minus the region size value remains for subsequent GETMAINs.

Note: For V=R jobs, the REGION parameter is more significant as a limiting value than are the limits set by IEFUSI. You can use the two factors together to control the region size for applications that must run V=R:

- Set the region size value where you want it, via IEFUSI.
- If a REGION parameter specification for a V=R job exceeds the region size value you have set, the job will not be initiated.

Coded Example of the Exit Routine

A sample IEFUSI exit routine is provided in SYS1.SAMPLIB in member IEEUSI. It sets the flag in the VSM parameter list indicating that it is controlling region sizes and limits instead of IEALIMIT. It is designed to perform the same processing as IEALIMIT in the control of region size and limit above and below the 16Mb line.

Chapter 33. IEFUSO — SYSOUT Limit Exit

Topics for This Exit Appear as Follows:

IEFUSO receives control from the job entry subsystem when the number of records written to an output data set exceeds the output limit for that data set. If the output limit is exceeded and your installation does not supply an IEFUSO exit routine, the job entry subsystem cancels the job. A return code from this exit indicates that the job is to be terminated or that the job should continue processing with a new output limit.

You can use IEFUSO to:

- Allow the job step to abend when a data set has exceeded its output limit.
- Inform the operator when a job exceeds its output limit for a data set and let the job continue processing.
- Extend output limits at the data set level for selected jobs.
- Keep a record of jobs that exceed output limits.
- Vary the handling of exceeding the output limit for different types of data sets or different types of jobs, such as teleprocessing, test, or production jobs.

Defining the Exit in SMFPRMxx

To allow the system to invoke IEFUSO, define the exit in the SMF parmlib member (SMFPRMxx). Specify IEFUSO on the EXITS option of the SUBSYS parameter for the STC subsystem. If your installation chooses not to define a SUBSYS parameter for STC, you can specify IEFUSO on the EXITS option of the SYS parameter.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFUSO installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFUSO or SYSyyy.IEFUSO. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the

EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

To define IEFUSO to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for this exit, you need only define IEFUSO in SMFPRMxx.

If you do not associate any exit routines with exit IEFUSO in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFUSO).

If you associate exit routines with this exit in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFUSO receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 1 from JES3 and key 0 from JES2.
- In AMODE 31 (JES2) and in AMODE 24 or 31 (JES3). For the valid combinations of AMODE and RMODE, see *z/OS MVS Programming: Assembler Services Guide*.

Exit Recovery

IBM strongly recommends that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of IEFUSO.

An ESTAE-type recovery routine is set up by the module that calls IEFUSO; the recovery routine, if it gets control, will allow the job to continue processing if the exit routine abnormally ends.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

IEFUSO receives control from the job entry subsystem when the output limit for a data set is exceeded. The output limit is specified by the OUTLIM parameter on the DD statement defining the output data set. Note that the OUTLIM parameter limits output only to spooled data sets. See *z/OS MVS JCL User's Guide* and *z/OS MVS JCL Reference* for a description of this parameter.

IEFUSO has the information in the SMF common exit parameter area available to it. When the exit routine completes its processing, it sets a return code in register 15. This value indicates whether the job entry subsystem (JES) is to cancel the job or allow it to continue processing with an increased output limit specified by the exit routine. The value for the increment to be applied to the output limit is placed in register 1.

Programming Considerations

SMF provides a replaceable module for each SMF exit routine. If an installation includes IEFUSO, certain programming standards must be followed:

- The exit routine must follow standard linkage conventions.
- Code the exit routine reenterable and refreshable.
- If IEFUSO is entered for a foreground job, it cannot access installation-defined data sets. If the exit routine is entered for a background job, it cannot write to installation-defined data sets.

IEFUSO sets a return code in register 15 to indicate whether or not processing is to continue with a new output limit. The value of the increment to the output limit is placed in register 1. If you specify with the return code that the limit is to be increased (register 15=4), but you do not increase the limit (register 1=0), then IEFUSO will receive control again when the next record is written to the output data set.

Note: For information about changing output limits at the job level for started tasks, refer to the following:

- For JES2, Exit 9 - Job Output Overflow, see *z/OS JES2 Installation Exits*.
- For JES3, Exit 29 - Examine the Accounting Information, see *z/OS JES3 Customization*.

Macro Instructions and Restrictions: When issuing a WTOR macro, specify LONG=YES on the WAIT macro. Do not use a WTO with a routing code of 11 to send a message to the JESYSMSG data set for started tasks or TSO users.

Entry Specifications

The job entry subsystem provides IEFUSO with addressability to the SMF common exit parameter area.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Contains the address of the parameter list (See Table 3 on page 156)
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEFUSO

Parameter Descriptions: Register 1 points to the following address:

Word 1
The address of the common exit parameter area (see "Common Exit Parameter Area" on page 155).

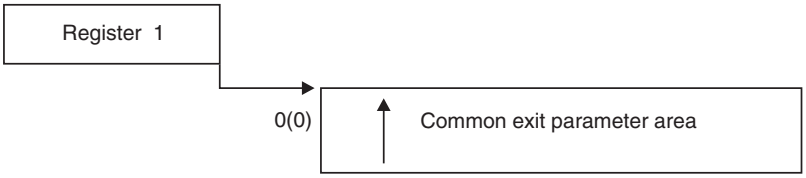

Figure 19. IEFUSO Input Parameter Structure

Return Specifications

A return code from IEFUSO indicates whether or not job processing is to continue.

If you associate multiple exit routines with IEFUSO, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFUSO, and any of those exit routines return with a value other than 4, job processing will not continue.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0 Restored to contents at entry

1 One of the following values:
- If R15=4, R1 contains the increment to the output limit.
- If R15 not=4, R1 restored to contents at entry.

2-14 Restored to contents at entry

15 One of the following return codes:

Return Code
Explanation

Value of 4
Continue processing the job. Increase the output limit by the value in register 1.

Value other than 4
Cancel the job.

Chapter 34. IEFUTL — Time Limit Exit

Topics for This Exit Appear as Follows:

IEFUTL receives control from the system when one of the following time limits expires:

- Job processor time limit (from the JOB statement)
- Step processor time limit (from the EXEC statement or the default from the job entry subsystem)
- Continuous wait time limit for the job (from the SMFPRMxx JWT parameter). "Continuous wait time" is defined as time spent waiting while the application program is in control. For example, the time required to recall a data set from HSM Migration Levels 1 or 2 and/or the time required to mount a tape is counted towards the job's continuous wait time if the allocation of the data set is dynamic (that is, issued while the application program is running) while the time required for those activities is not counted toward the job's continuous wait time if the allocation is static (that is, for a DD statement). This is because static allocations occur under the initiator rather than under the application program.

If one of the above time limits is exceeded and your installation does not supply an IEFUTL exit routine, the system cancels the job. A return code from this exit indicates whether the job step is to be terminated or processing should continue with a new time limit.

You can use IEFUTL to:

- Allow the job step to abend.
- Inform the operator that a job has exceeded its continuous-wait-time limits.
- Extend processor time limits for selected jobs.
- Extend the wait time limit within a job.
- Keep a record of time limit expirations.
- Vary the handling of time limit expirations for different types of jobs, such as teleprocessing, test, or production jobs.

Defining the Exit in SMFPRMxx

In the SMF parmlib member (SMFPRMxx), specify IEFUTL on the EXITS option of either the SYS or SUBSYS parameters, depending on the scope of work (system-wide or subsystem-wide) the exit is to affect.

If you use the SUBSYS option, the system invokes the IEFUTL routine only for work running under the subsystems you specify on SUBSYS. If you use the SYS option, the system invokes the IEFUTL routine for work running under any SMF-defined subsystem, such as JES2, JES3, STC, ASCH, OMVS, or TSO.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFUTL installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFUTL or SYSyyy.IEFUTL. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

To define IEFUTL to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for IEFUTL, you need only define this exit in SMFPRMxx.

If you do not associate any exit routines with exit IEFUTL in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFUTL).

If you associate exit routines with this exit in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFUTL receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 24 or 31. For the valid combinations of AMODE and RMODE, see *z/OS MVS Programming: Assembler Services Guide*.
- In IRB mode as an asynchronous exit.

Exit Recovery

IBM strongly recommends that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of IEFUTL.

An ESTAE-type recovery routine is set up by the module that calls IEFUTL. The recovery routine, if it gets control, will allow the job to continue processing if the exit routine abnormally ends.

Because the purpose of IEFUTL is to decide whether a step that has exceeded its allotted processor time or wait time should be abnormally terminated (the system default) or allowed to continue, be aware that the job WILL abend with the S322 or S522 it would have received if the exit routine had never gotten control.

Exit Routine Processing

Processor time is collected in two categories: execution under TCBs and execution under SRBs. The limiting function of IEFUTL applies only to such time under TCBs.

IEFUTL receives control when a time limit has expired. While IEFUTL is running, the application program continues to execute. (Note that this represents a change from MVS/XA and MVS/ESA up through release 4, wherein the system would suspend execution of an application program while IEFUTL was running.) This means, for example, that if IEFUTL issues a WTOR asking the operator whether to allow a job to continue executing or to cancel it, the job will, in fact, continue to execute while IEFUTL is waiting for the answer to the WTOR.

Depending on whether the expired time limit is a job, step, or continuous wait time limit, you can use the information in the SMF common exit parameter area to determine if processing should continue. The value of the time limit extension is either in seconds or in timer units, where 1 second = 38400 timer units. The smallest time extension granted is 2^{20} microseconds or 1.048576 seconds.

The time limit for the execution of a job step is specified by the job entry subsystem or by parameters on job control statements.

- If a job time limit is not specified on the JOB statement, the time limit for each job step is the value specified for the TIME=parameter on the EXEC statement, or the default value from the job entry subsystem.
- If a job time limit is specified on the JOB statement, the time limit for each job step is the remaining job time or the job step time limit (from the TIME=parameter or the job entry subsystem default), whichever is smaller.

You can extend execution time and wait time only within a step. Each extension resets the limit for the entire step to the extension value you specify.

The step execution and the wait time limits are re-initialized to the system default values at the beginning of each job step. Thus, unused extended execution time from one job step is not carried over for the next step.

An installation-written IEFUTL exit routine should control the number of extensions for a given step to prevent looping. It can record the expiration in the SMF data set or write a message to the console; however, in doing so, a system interlock could occur. (See “Programming Considerations” on page 230.)

z/OS UNIX MVS Address Space Processing: The following applies to address spaces that are z/OS UNIX MVS processes:

- If the time expiration is for a job or step time limit and IEFUTL exit processing indicates that the job step should be terminated, the system takes the following action:
  1. The system sends a SIGXCPU signal to the z/OS UNIX MVS process, and grants a small time extension to allow for SIGXCPU processing to occur. Applications can catch SIGXCPU signals and perform an orderly cleanup of the job or reset the CPU limit to a larger value.

     Note: Processes can use the z/OS UNIX MVS **setrlimit** callable service to control CPU resource consumption. For information about z/OS UNIX MVS signals and the **setrlimit** callable service, see *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.
  2. If the time extension for SIGXCPU processing expires, the system sends a SIGKILL signal to the z/OS UNIX MVS process and grants a small time extension to allow SIGKILL processing to occur. Applications cannot catch or ignore SIGKILL signals. SIGKILL signals provide for a more orderly shutdown of the application than an abend of the job step.
  3. If the time extension for SIGKILL processing expires, the system abends the job step.
- If the time expiration is for a job or step time limit and IEFUTL exit processing indicates that a time extension should be granted, the system grants the time extension.
- If the time expiration is for a continuous wait time limit and IEFUTL exit processing indicates that the job step should be terminated, the system abends the job step.
- If the time expiration is for a continuous wait time limit and IEFUTL exit processing indicates that a time extension should be granted, the system grants the time extension.

Programming Considerations

IEFUTL must be reenterable and refreshable, because PLPA pages are stolen. That is, they can be paged in but not paged out, and subsequent page-ins overlay any code changes.

IEFUTL can perform dynamic allocations and write to installation-defined data sets. In foreground jobs, data sets are allocated dynamically. For background jobs, you can either allocate data sets dynamically or you can pre-define (pre-allocate) a data set with a DD statement in the initiator cataloged procedure.

When issuing a WTOR macro, specify LONG=YES on the WAIT macro. Do not use a WTO with a routing code of 11 to send a message to the JESYSMSG data set for started tasks or TSO users. To provide a consistent environment for accessing and allocating data sets across calls to SMF exits for the duration of a job or task, IEFUTL receives control with the initiator's JSCB active.

If IEFUTL enqueues on any resource that the job task or any of its subtasks is enqueued on, the initiator ends abnormally. IEFUTL can, however, determine if a particular resource is held before issuing an ENQ (or invoking an SVC that issues an ENQ) by issuing an ENQ macro with RET=TEST. The macro must also specify the major and minor resource names in the QNAME and RNAME parameters. For example:

```
ENQ(QNAME,RNAME,E,3,SYSTEM),RET=TEST
```

Because SMF exits must be reentrant, be sure to use the execute form of the macro.

For more information on the ENQ macro, see *z/OS MVS Programming: Assembler Services Guide* and *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

Entry Specifications

IEFUTL is passed the address of the SMF common exit parameter area and the type of time limit that expired to determine whether processing should continue.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register
Contents

0 A binary code to indicate why the exit is taking control:

Binary Code **Explanation**

0 The processor time limit for the job expired

4 The processor time limit for the step expired

8 The continuous wait time limit for the job expired

1 Address of the parameter list

2-12 Not applicable

13 Register save area

14 Return address

15 Entry point address of IEFUTL

Parameter Descriptions: Register 1 points to the following list of addresses:

Word 1
The address of the common exit parameter area (see "Common Exit Parameter Area" on page 155).

Word 2
The address of a 4-character area that contains the name of the subsystem for the job being processed. Examples:

- ASCH, JES2, or JES3 - indicates the name of the subsystem that selected the job
- OMVS - indicates a forked or spawned address space
- STC - indicates a started task
- TSO - indicates a time sharing option task
- The jobname - used if it is four or fewer characters and none of the above apply

Note: The high-order bit is set in the address of the last parameter to indicate the end of the parameter list.

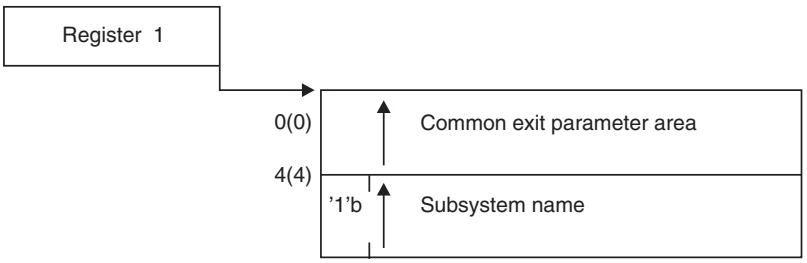

Figure 20. IEFUTL Input Parameter Structure

Return Specifications

If IEFUTL returns a code that indicates that processing should continue, then the time extension to be applied is returned in register 1.

If you associate multiple exit routines with IEFUTL, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFUTL, and any of those exit routines return with a value other than 4 or 8, job processing is cancelled.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0 Not applicable

1 Contains one of the following time extensions:
- If R15=4, R1 contains a time extension in timer units.
- If R15=8, R1 contains a time extension in seconds.

2-14 Not applicable

15 One of the following return codes:

Return Code
Explanation

4 Job processing should be continued with a time extension in timer units.

8 Job processing should be continued with a time extension in seconds.

Other value
Job processing should be cancelled.

Coded Example of the Exit Routine

Sample IEFUTL exit routines are provided in SYS1.SAMPLIB in members SMFEXITS and IEEUTL. This routine terminates a job if either the job processor time limit or the job step processor time limit has been exceeded. If the continuous wait time limit for the job has been exceeded, the routine extends the limit twice; on the third entry for exceeding the continuous wait time limit, the routine cancels the job.

The continuous wait time limit is not an accumulation of all the time the task spends waiting, but rather a single continuous wait period that exceeds the specified limit.

Each time the routine is invoked for exceeding the continuous wait time limit, the routine writes a record to the SMF data set describing the action taken.

The IEEUTL exit routine is the SMF time limit exit. Since TSO sessions are handled like batch jobs, the SMF job wait limit and CPU time limits are enforced. This routine checks to see if the address space is a TSO session, and, if the CPU time is exceeded, issues a warning message to the terminal and grants a one-minute extension before the session is cancelled.

Chapter 35. IEFU29 — SMF Dump Exit

Topics for This Exit Appear as Follows:

The SMF dump exit IEFU29 is invoked when the current recording data set cannot hold any more records, since the SMF writer routine automatically switches recording from the active SMF data set to an empty SMF data set. This exit is also invoked when the writer switches recording data sets as a result of the SWITCH SMF command. A return code from this exit routine indicates whether a message that the SMF data set requires dumping should be suppressed or not.

You can use IEFU29 to:

- Issue the WTO macro to request that the operator start the dump program.
- Initiate the dump program by submitting a job request to an internal reader.

IEFU29 will also be invoked during SMF initialization for alternate data sets that are not empty.

Defining the Exit in SMFPRMxx

To allow the system to invoke IEFU29, define the exit in the SMF parmlib member (SMFPRMxx). Specify IEFU29 on the EXITS option of the SUBSYS parameter for the STC subsystem. If your installation chooses not to define a SUBSYS parameter for STC, you can specify IEFU29 on the EXITS option of the SYS parameter.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFU29 installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFU29 or SYSyyy.IEFU29. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

To define IEFU29 to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for this exit, you need only define IEFU29 in SMFPRMxx.

If you do not associate any exit routines with exit IEFU29 in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFU29).

If you associate exit routines with this exit in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFU29 receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 24 or 31: When the SMF data set name is in the SYS1.MANx format.
- In AMODE 31: When the SMF data set name is in a format other than SYS1.MANx.
- In the SMF address space.

Exit Recovery

IBM strongly recommends that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of IEFU29.

An ESTAE-type recovery routine is set up by the module that calls IEFU29; the recovery routine, if it gets control, will prevent SMF from ending if the exit routine abnormally ends.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

While the SMF writer records on one data set, the others can be written out (or cleared). As long as one inactive data set is empty when the active data set becomes full, the SMF writer continues to record. If none is available, no further recording takes place.

The SMF dump exit receives control from the SMF writer when an SMF data set becomes full. IEFU29 returns a code specifying that the control program either

issue or suppress the dump message (IEE362A, IEE362I, IEE391A or IEE392I).

Programming Considerations

IEFU29 must be reenterable and refreshable because PLPA pages are stolen. That is, they can be paged in but not paged out, and subsequent page-ins overlay any code changes.

Note: IEFU29 runs in the SMF address space which runs under the MSTR subsystem rather than JES. Therefore, jobs submitted to the internal reader from IEFU29 will run under MSTR rather than JES. To have a job run under JES, you must request that the job run on JES rather than the MSTR subsystem. For more information on submitting a job request to an internal reader, see *z/OS MVS Using the Subsystem Interface*.

Macro Instructions and Restrictions: IEFU29 can issue the WTOR macro (for example, to request the operator to start the dump program). When issuing the WTOR macro, specify LONG=YES on the WAIT macro.

IEFU29 cannot use the SMFWTM or SMFEWTM macro to write records to the SMF data set.

Entry Specifications

SMF passes to IEFU29 the address of the name of the SMF data set that requires dumping; see Figure 21.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the name of the SMF data set
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEFU29

Parameter Descriptions: Register 1 points to the following address:

Word 1
The address of the 44-character field that contains the name of the SMF data set that requires dumping. This field is left-justified and padded on the right with blanks.

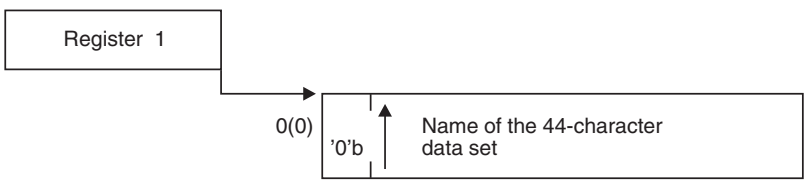

Figure 21. IEFU29 Input Parameter Structure

Return Specifications

A return code from IEFU29 indicates whether the dump message (IEE362A, IEE362I, IEE391A or IEE392I) is to be issued.

If you associate multiple exit routines with IEFU29, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFU29, and any of those exit routines return with a value of 4, the dump message (IEE362A, IEE362I, IEE391A or IEE392I) is not issued.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-14 Not applicable

15 One of the following return codes:

Return Code
Explanation

4 Indicates that the dump message (IEE362A, IEE362I, IEE391A or IEE392I) is not to be issued.

Other than 4
The message is to be written.

Coded Example of the Exit Routine

IEFU29 exit routine is entered when an SMF dataset is switched. On entry, the SMF data set name which was recording before the switch is passed to the routine. Depending on which data set name is passed, the sample exit uses SVC 34 to issue a START DUMPXY,DSNAME=dsname operator command. Using a WTO, this exit routine informs the operator that the command has been issued, the DUMPXY procedure saves the data in a data set and clears the SMF recording data set.

| Figure 22 on page 239 is an example of a SMF Dump Exit.
|

```
D SMF
IFA714I 18.35.11 SMF STATUS
          LOGSTREAM NAME BUFFERS STATUS
         A-IFASMF.STRIPE.TYPDFLT 37485 CONNECTED
         A-IFASMF.#@$#PLEX.TYPRMF 12464 CONNECTED
I SMF
IFA705I SWITCH SMF PROCESS HAS SYNCHRONIZED THE BUFFERED LOGSTREAM RECORDS.
* * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                 *
* SMF LOGSTREAM SWITCH EXIT                       *
*                                                 *
* * * * * * * * * * * * * * * * * * * * * * * * * *
         LsName(1): IFASMF.STRIPE.TYPDFLT
         LsName(2): IFASMF.#@$#PLEX.TYPRMF
```

Figure 22. Example: SMF Dump Exit

Chapter 36. IEFU29L — SMF Log Stream Dump Exit

Topics for This Exit Appear as Follows:

The SMF dump exit IEFU29L allows you to initiate the archiving of SMF data from a log stream. IEFU29L is invoked using the SWITCH SMF command.

Note that you can use the system logger to manage how long you retain the SMF log stream data and to automatically offload the log stream data to VSAM linear DASD data sets, so you might not need to use IEFU29L to drive the archiving of the SMF log stream data. See "Managing Log Data: How Much? For How Long?"in *z/OS MVS Setting Up a Sysplex*.

Use IEFU29L to perform the following tasks:

- Issue the WTO macro to request that the operator start the SMF log stream dump program, IFASMFDL.
- Initiate the dump program by submitting a job request to an internal reader.

Defining the Exit in SMFPRMxx

To allow the system to invoke IEFU29L, define the exit in the SMF parmlib member (SMFPRMxx). Specify IEFU29L on the EXITS option of the SUBSYS

parameter for the STC subsystem. If your installation chooses not to define a SUBSYS parameter for STC, you can specify IEFU29L on the EXITS option of the SYS parameter.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFU29L installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFU29L or SYSyyy.IEFU29L. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

To define IEFU29L to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for this exit, you only need to define IEFU29L in SMFPRMxx.

If you do not associate any exit routines with exit IEFU29L in PROGxx, by default, the system uses the exit routine name that matches the exit name (IEFU29L).

If you associate exit routines with this exit in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times that the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error.
- The system allows a retry, that is, the recovery routine is entered with the bit SDWACLUP off.

By default, the system does not disable the exit routine.

Exit Routine Environment

IEFU29L receives control in the following environment:

- Enabled for interruptions.
- In supervisor state with PSW key 0.
- In AMODE 24 or 31.
- In the SMF address space.

Exit Recovery

IBM suggests that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of IEFU29L.

An ESTAE-type recovery routine is set up by the module that calls IEFU29L; if the recovery routine gets control, it will prevent SMF from ending when the exit routine ends abnormally.

Whether the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

The SMF dump exit receives control from the SMF writer when the SWITCH SMF command is issued to dump data from an SMF log stream.

Programming Considerations

IEFU29L must be reenterable and refreshable because PLPA pages are stolen. That is, they can be paged in but not paged out, and subsequent page-ins overlay any code changes.

Note: IEFU29L runs in the SMF address space that runs under the MSTR subsystem rather than JES. Therefore, jobs submitted to the internal reader from IEFU29L will run under MSTR rather than JES. To have a job run under JES, you must request that the job run on JES rather than the MSTR subsystem. For more information about submitting a job request to an internal reader, see *z/OS MVS Using the Subsystem Interface*.

Macro Instructions and Restrictions: IEFU29L can issue the WTOR macro (for example, to request the operator to start the dump program). When you issue the WTOR macro, specify LONG=YES on the WAIT macro.

IEFU29L cannot use the SMFWTM or SMFEWTM macro to write records to the SMF log stream.

Entry Specifications

SMF passes to IEFU29L a parameter list mapped by IEFU29LM, which contains the name of the log stream.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of a pointer to a parameter list of IEFU29L, mapped by the IFAU29LM control block
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEFU29L

Parameter Descriptions: When IEFU29L is driven, control block IEFU29LM will receive Register 1, which points to the following address. Figure 23 on page 242 shows an example of input parameter structure.

Word 1
An array of pointers, each of which point to U29L_PARM mapped in IFAU29LM. The high order bit of the pointer signifies that it is the last one.

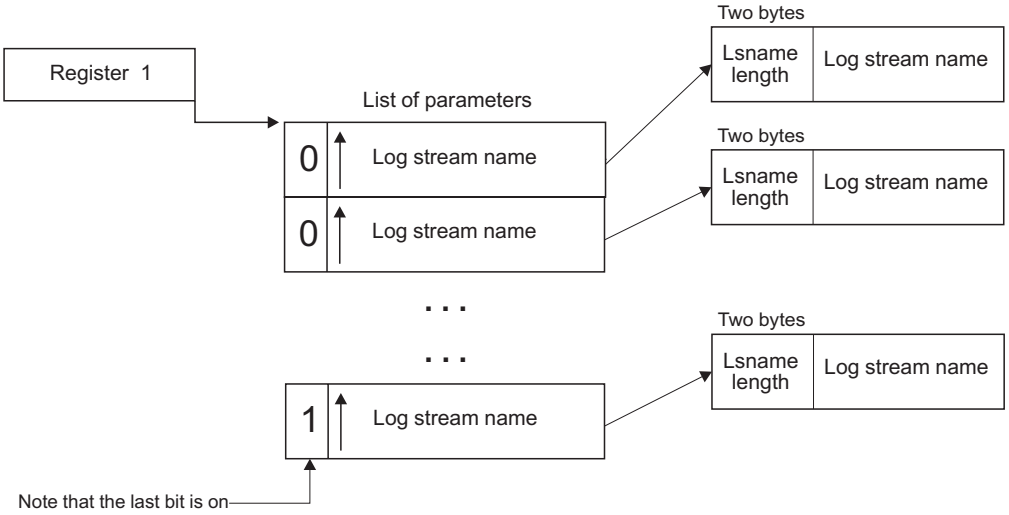

Figure 23. IEFU29L Input Parameter Structure

Return Specifications

No return information is expected.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-15 Not applicable

Coded Example of the Exit Routine

A sample IEFU29L exit is provided in SYS1.SAMPLIB in member IEEU29. The sample exit consists of an Assembler stub that calls a REXX program, passing the names of the log streams as parameters. The sample REXX program returns the log stream names to the console and can be modified as necessary.

Chapter 37. IEFU83 — SMF Record Exit

Topics for This Exit Appear as Follows:

The SMF record exit IEFU83 receives control when the caller invokes either:

- The SMFWTM macro.
- The SMFEWTM macro and specifies BRANCH=NO.

After exit routine processing, IEFU83 returns a code that specifies whether SMF is to write the SMF record to the SMF data set. IEFU83 does not receive control for records whose writing has been suppressed either because of a system failure or because of options selected at IPL time or via the SET SMF command.

You can use IEFU83 to:

- Select or suppress those records to be written to the SMF data set. For example, an installation with a large TSO/E account might want to suppress the SMF dynamic DD records (type 40).
- Check resource use during a specific interval. For example, select records during the peak workload period.

Defining the Exit in SMFPRMxx

In the SMF parmlib member (SMFPRMxx), specify IEFU83 on the EXITS option of either the SYS or SUBSYS parameters, depending on the scope of work (system-wide or subsystem-wide) the exit is to affect.

If you use the SUBSYS option, the system invokes the IEFU83 routine only for work running under the subsystems you specify on SUBSYS. If you use the SYS option, the system invokes the IEFU83 routine for work running under any SMF-defined subsystem, such as JES2, JES3, STC, ASCH, OMVS, or TSO.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFU83 installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFU83 or SYSyyy.IEFU83. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

To define IEFU83 to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for this exit, you need only define IEFU83 in SMFPRMxx.

If you do not associate any exit routines with exit IEFU83 in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFU83).

If you associate exit routines with this exit in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system disables the exit routine after two consecutive abends.

Exit Routine Environment

IEFU83 receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31.

Exit Recovery

If IEFU83 abnormally terminates, SMF, in most cases, does not terminate. SMF deactivates the exit and then issues message CSV430I to the operator. If the exit performs a critical function, the operator can issue a SET SMF or SETSMF command to terminate recording. Otherwise, SMF recording continues but bypasses the installation exit routine.

IBM strongly recommends that you set up an ESTAEX recovery routine to handle errors that might occur during the execution of your exit routine.

An ESTAE-type recovery routine is set up by the module that calls IEFU83; the recovery routine, if it gets control, will prevent SMF from ending if the exit routine abnormally ends. If this recovery routine gets control on two consecutive invocations of the exit, SMF requests that the exit routine be marked inactive,

preventing any further invocations of that exit routine. The system issues message CSV430I, naming the exit and the exit routine.

If the exit performs a critical function, the operator can issue a SET PROG or SETPROG MODIFY command to change the status of the exit to active. This should be done only if you have corrected the program, or if you know the error conditions are transient.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

Each SMF record is passed to an installation exit (either IEFU83, IEFU84, or IEFU85) before it is written to the SMF data set. If you use the SMFWTM macro, or if you specify BRANCH=NO on the SMFEWTM macro, SMF invokes installation exit IEFU83. If you use the SMFEWTM macro and specify BRANCH=YES, SMF invokes installation exit IEFU84.

IEFU83 places a return code in register 15 before returning control; the code indicates whether the record should be written to the SMF data set.

Programming Considerations

IEFU83 must be reenterable and refreshable, because PLPA pages are stolen. That is, they can be paged in but not paged out, and subsequent page-ins overlay any code changes.

If IEFU83 is entered for a foreground job, it cannot access installation-defined data sets. If the exit is entered for a background job, it cannot write to installation-defined data sets.

The addresses of the user communication and user identification fields of the common exit parameter area (a copy of the first 36 bytes of the JMR) are not passed to the IEFU83 exit routine. To obtain these addresses, the exit routine can follow pointers from the PSA to the TCB to the JMR. In the PSA, the PSATOLD points to the TCB. In the TCB, the TCBTCT field points to the TCT. In the TCT, the TCTJMR field points to the JMR. The JMR and the PSA are mapped by macro IEFJMR and IHAPSA, respectively. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the JMR and the mappings of the TCB and the TCT.

APPC/MVS Consideration: You might want your exit routine to perform specific processing for work initiated by the APPC/MVS transaction scheduler. To enable your exit routine to determine when it has been invoked during this type of processing, have the routine check the OUCBSUBN field in the OUCB control block. For APPC/MVS TPs, this field is set to 'ASCH' (EBCDIC).

The exit routine can access OUCBSUBN by chaining through pointers from the PSA to the ASCB to the OUCB. In the PSA, the PSAAOLD points to the ASCB. In the ASCB, the ASCBOUCB field points to the OUCB.

The ASCB, OUCB, and the PSA are mapped by macros IHAASCB, IRAOUCB, and IHAPSA, respectively. The mappings are described in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Macro Instructions and Restrictions: When issuing a WTOR macro, specify LONG=YES on the WAIT macro.

IEFU83 cannot use the SMFWTM or SMFEWTM macro to write to the SMF data set.

Entry Specifications

SMF passes to IEFU83 the address of a fullword that points to the SMF record.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the parameter list.
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEFU83

Parameter Descriptions: Register 1 points to the following address:

Word 1
The address of the record that SMF is to write. The first four bytes of this record are the record descriptor word (RDW). See *z/OS MVS System Management Facilities (SMF)* for a description of the RDW.

Figure 24. IEFU83 Input Parameter Structure

Return Specifications

A return code from IEFU83 indicates whether the current SMF record is to be suppressed.

If you associate multiple exit routines with IEFU83, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFU83, and any of those exit routines return with a value of 4, the current SMF record is suppressed.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

1 Not applicable

2-14 Not applicable

15 One of the following return codes:

Return Code
Explanation

4 SMF is not to write the record to the SMF data set.

Other than 4
SMF is to write the record to the SMF data set.

Chapter 38. IEFU84 — SMF Record Exit

Topics for This Exit Appear as Follows:

The SMF record exit IEFU84 receives control when the caller invokes the SMFEWTM macro, specifying BRANCH=YES. After exit routine processing, IEFU84 returns a code that specifies whether the SMF record should be written to the SMF data set. The SMFEWTM macro allows the issuer to branch directly to the SVC routine without issuing the SVC. IEFU84 does not receive control for records suppressed because of options selected at IPL time or via the SET SMF command.

You can use IEFU84 to:

- Select or suppress those records to be written to the SMF data set. For example, an installation with a large TSO/E account might want to suppress SMF records for all but a few selected TSO/E users.
- Check resource use during a specific interval. For example, select records during the peak workload period.
- Suppress some of the record type 30 subtypes.

Defining the Exit in SMFPRMxx

In the SMF parmlib member (SMFPRMxx), specify IEFU84 on the EXITS option of either the SYS or SUBSYS parameters, depending on the scope of work (system-wide or subsystem-wide) the exit is to affect.

If you use the SUBSYS option, the system invokes the IEFU84 routine only for work running under the subsystems you specify on SUBSYS. If you use the SYS option, the system invokes the IEFU84 routine for work running under any SMF-defined subsystem, such as JES2, JES3, STC, ASCH, OMVS, or TSO.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFU84 installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFU84 or SYSyyy.IEFU84. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

To define IEFU84 to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for this exit, you need only define IEFU84 in SMFPRMxx.

If you do not associate any exit routines with exit IEFU84 in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFU84).

If you associate exit routines with this exit in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system disables the exit routine after two consecutive abends.

Exit Routine Environment

IEFU84 receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31.
- In the address space of the task that issues the SMFEWTM BRANCH=YES macro.
- Can be locked or in SRB mode.

Exit Recovery

If IEFU84 abnormally terminates, SMF, in most cases, does not terminate. SMF marks the exit as not valid and issues message IEE952I to the operator. If the exit performs a critical function, the operator can issue a SET SMF or SETSMF command to terminate recording. Otherwise, SMF recording continues but bypasses the installation exit routine.

IBM strongly recommends that you set up an FRR recovery routine to handle errors that might occur during the execution of your exit routine. The FRR should use the EUT=YES option to handle errors that occur when the exit is called in unlocked task mode.

An FRR is set up by the module that calls IEFU84; the recovery routine, if it gets control, will prevent SMF from ending if the exit routine abnormally ends. If this recovery routine gets control on two consecutive invocations of the exit, SMF requests that the exit routine be marked inactive, preventing any further invocations of that exit routine. The system issues message CSV430I, naming the exit and the exit routine.

If the exit performs a critical function, the operator can issue a SET PROG or SETPROG MODIFY command to change the status of the exit to active. This should be done only if you have corrected the program, or if you know the error conditions are transient.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

Each SMF record is passed to an installation exit (either IEFU83, IEFU84 or IEFU85) before it is written to the SMF data set. If you use the SMFEWTM macro and specify BRANCH=YES, SMF invokes installation exit IEFU84. If you use the SMFWTM macro or if you specify BRANCH=NO on the SMFEWTM macro, SMF invokes installation exit IEFU83.

The SMFEWTM macro verifies that SMF recording is active and allows the issuer to branch directly to the SVC routine without issuing the SVC. BRANCH=YES causes the macro to generate a call to the subroutine that moves the data to the SMF buffer.

IEFU84 places a return code in register 15 before returning control; the code indicates whether the record is to be written to the SMF data set.

Programming Considerations

IEFU84 must be reenterable and refreshable, because PLPA pages are stolen. That is, they can be paged in but not paged out, and subsequent page-ins overlay any code changes.

IEFU84 cannot access installation-defined data sets.

The addresses of the user communication and user identification fields of the common exit parameter area (a copy of the first 36 bytes of the JMR) are not passed to the IEFU84 exit routine. To obtain these addresses, the exit routine can follow pointers from the ASXB to the TCB to the JMR. In the ASXB, the ASXBLTCB points to a chain of TCBs. The ASXB can be found from the field ASCBASXB in the ASCB which in turn can be found from the PSAAOLD in the PSA.

In the TCB, the TCBTCT field points to the TCT. In the TCT, the TCTJMR field points to the JMR. The JMR and the PSA are mapped by macros IEFJMR and IHAPSA, respectively. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the JMR, and *z/OS MVS Data Areas, Vol 5 (SSAG-XTLST)* for the mappings of the TCB and the TCT.

Note: If you use the ASXBLTCB, this would only be the LAST TCB that was attached in the address space, and would not necessarily be running, or even dispatchable. You would have determine that by looking at the dispatchability bits

in the TCB. Or you could start with the ASXBFTCB, which will point to the Region Control Task's TCB, and chain down from that using the TCBTCB field.

APPC/MVS Consideration: You might want your exit routine to perform specific processing for work initiated by the APPC/MVS transaction scheduler. To enable your exit routine to determine when it has been invoked during this type of processing, have the routine check the OUCBSUBN field in the OUCB control block. For APPC/MVS TPs, this field is set to 'ASCH' (EBCDIC).

The exit routine can access OUCBSUBN by chaining through pointers from the PSA to the ASCB to the OUCB. In the PSA, the PSAAOLD points to the ASCB. In the ASCB, the ASCBOUCB field points to the OUCB.

The ASCB, OUCB, and the PSA are mapped by macros IHAASCB, IRAOUCB, and IHAPSA, respectively. The mappings are described in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Macro Instructions and Restrictions: When issuing a WTOR macro, specify LONG=YES on the WAIT macro.

IEFU84 cannot use the SMFWTM or SMFEWTM macro to write to the SMF data set.

Because IEFU84 might be locked or in SRB mode, the exit routine cannot issue any SVCs. IEFU83 may be given the alias name IEFU84, if IEFU83 can run locked or SRB mode.

Entry Specifications

SMF passes to IEFU84 the address of a fullword that points to the SMF record.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	**Contents**
0	Not applicable
1	Address of the parameter list
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEFU84

Parameter Descriptions: Register 1 points to the following address:

Word 1
The address of the record that SMF is to write. The first four bytes of this record are the record descriptor word (RDW). See *z/OS MVS System Management Facilities (SMF)* for a description of the RDW.

Figure 25. IEFU84 Input Parameter Structure

Return Specifications

A return code from IEFU84 indicates whether the current SMF record is to be suppressed.

If you associate multiple exit routines with IEFU84, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFU84, and any of those exit routines return with a value of 4, the system does not write the record to the SMF data set.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-14 Same at entry

15 One of the following return codes:

Return Code
Explanation

Value of 4
SMF is not to write the record to the SMF data set.

Other than 4
SMF is to write the record to the SMF data set.

Chapter 39. IEFU85 — SMF Record Exit

Topics for This Exit Appear as Follows:

The SMF record exit IEFU85 is used for a caller who is running in cross memory mode. The exit routine receives control when the caller invokes the SMFEWTM macro, specifying BRANCH=YES and MODE=XMEM and when ASCB does not equal PSAAOLD (the home primary ASID). After exit routine processing, IEFU85 returns a code that specifies whether the SMF record should be written to the SMF data set. The SMFEWTM macro allows the issuer to branch directly to the SVC routine without issuing the SVC. IEFU85 does not receive control for records that are suppressed because of options selected at IPL time or via the SET SMF command.

You can use IEFU85 to:

- Select or suppress those records to be written to the SMF data set. For example, an installation with a large TSO/E account might want to suppress SMF records for all but a few selected TSO/E users.
- Check resource use during a specific interval. For example, select records during the peak workload period.

Defining the Exit in SMFPRMxx

In the SMF parmlib member (SMFPRMxx), specify IEFU85 on the EXITS option of either the SYS or SUBSYS parameters, depending on the scope of work (system-wide or subsystem-wide) the exit is to affect.

If you use the SUBSYS option, the system invokes the IEFU85 routine only for work running under the subsystems you specify on SUBSYS. If you use the SYS option, the system invokes the IEFU85 routine for work running under any SMF-defined subsystem, such as JES2, JES3, STC, ASCH, OMVS, or TSO.

For more information about coding the EXITS option, see the description of SMFPRMxx in *z/OS MVS Initialization and Tuning Reference*.

Controlling the Exit Routine Through the Dynamic Exits Facility

IBM has defined the IEFU85 installation exit to the dynamic exits facility. You can refer to the exit by the name SYS.IEFU85 or SYSyyy.IEFU85. See the description of the SMFPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference* for an explanation of the naming conventions for SMF exit routines. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

To define IEFU85 to the dynamic exits facility, you must specify the exit in both PROGxx and SMFPRMxx. The system does not call the exit if it is defined in PROGxx only. If you do not plan to use the dynamic exits facility for this exit, you need only define IEFU85 in SMFPRMxx.

If you do not associate any exit routines with exit IEFU85 in PROGxx, the system defaults to using the exit routine name that matches the exit name (IEFU85).

If you associate exit routines with this exit in PROGxx, the system does not use the default exit routine. If you need the default exit routine, you should explicitly add it to PROGxx.

You can use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM parameter of the SETPROG EXIT operator command to limit the number of times the exit routine abnormally ends before it becomes inactive. An abend is counted when both of the following conditions exist:

- The exit routine does not provide recovery, or the exit routine does provide recovery but percolates the error
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system disables the exit routine after two consecutive abends.

Exit Routine Environment

IEFU85 receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31.
- In the address space of the task that issues the SMFEWTM BRANCH=YES MODE=XMEM macro.
- May be locked.
- In cross memory mode.

Exit Recovery

If IEFU85 abnormally terminates, SMF, in most cases, does not terminate. SMF marks the exit as not valid and issues message IEE952I to the operator. If the exit performs a critical function, the operator can issue a SET SMF or SETSMF command to terminate recording. Otherwise, SMF recording continues but bypasses the installation exit routine.

A functional recovery routine (FRR) is set up by IEEMB830 (the SVC 83 routine) because the caller is in cross memory mode.

IBM strongly recommends that you set up an FRR recovery routine to handle errors that might occur during the execution of your exit routine.

An FRR is set up by the module that calls IEFU85; the recovery routine, if it gets control, will prevent SMF from ending if the exit routine abnormally ends. If this recovery routine gets control on two consecutive invocations of the exit, SMF requests that the exit routine be marked inactive, preventing any further invocations of that exit routine. The system issues message CSV430I, naming the exit and the exit routine.

If the exit performs a critical function, the operator can issue a SET PROG or SETPROG MODIFY command to change the status of the exit to active. This should be done only if you have corrected the program, or if you know the error conditions are transient.

Whether or not the exit routine continues to be invoked depends on the abend processing of the dynamic exits facility.

Exit Routine Processing

Each SMF record is passed to an installation exit (either IEFU83, IEFU84, or IEFU85) before it is written to the SMF data set. If you use the SMFEWTM macro and specify BRANCH=YES MODE=XMEM, SMF invokes installation exit IEFU85.

The SMFEWTM macro verifies that SMF recording is active and allows the issuer to branch directly to the SVC routine without issuing the SVC. BRANCH=YES causes the macro to generate a call to the subroutine that moves the data to the SMF buffer.

IEFU85 places a return code in register 15 before returning control; the code indicates whether the record is to be written to the SMF data set.

Programming Considerations

IEFU85 must be reenterable and refreshable, because PLPA pages are stolen. That is, PLPA pages can be paged in but not paged out, and subsequent page-ins overlay any code changes.

IEFU85 cannot access installation-defined data sets.

Because IEFU85 is running in cross-memory mode, the exit routine cannot issue any SVCs.

The addresses of the user communication and user identification fields of the common exit parameter area (a copy of the first 36 bytes of the JMR) are not passed to the IEFU85 exit routine. To obtain these addresses, the exit routine can follow pointers from the PSA to the TCB to the JMR. In the PSA, the PSATOLD points to TCB. In the TCB, the TCBTCT field points to the TCT. In the TCT, the TCTJMR field points to the JMR. The JMR and the PSA are mapped by macros IEFJMR and IHAPSA, respectively. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the JMR, and *z/OS MVS Data Areas, Vol 5 (SSAG-XTLST)* for the mappings of the TCB and the TCT.

APPC/MVS Consideration: You might want your exit routine to perform specific processing for work initiated by the APPC/MVS transaction scheduler. To enable

your exit routine to determine when it has been invoked during this type of processing, have the routine check the OUCBSUBN field in the OUCB control block. For APPC/MVS TPs, this field is set to 'ASCH' (EBCDIC).

The exit routine can access OUCBSUBN by chaining through pointers from the PSA to the ASCB to the OUCB. In the PSA, the PSAAOLD points to the ASCB. In the ASCB, the ASCBOUCB field points to the OUCB.

The ASCB, OUCB, and the PSA are mapped by macros IHAASCB, IRAOUCB, and IHAPSA, respectively. The mappings are described in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Macro Instructions and Restrictions: IEFU85 can issue only macros that can run in cross-memory mode. See the macro summary in *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information on macros that can run in cross-memory mode.

IEFU85 cannot use the SMFWTM or SMFEWTM macro to write to the SMF data set.

Entry Specifications

SMF passes to IEFU85 the address of a fullword that points to the SMF record.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the parameter list.
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of IEFU85

Parameter Descriptions: Register 1 points to the following address:

Word 1
The address of the record that SMF is to write. The first four bytes of this record are the record descriptor word (RDW). See *z/OS MVS System Management Facilities (SMF)* for a description of the RDW.

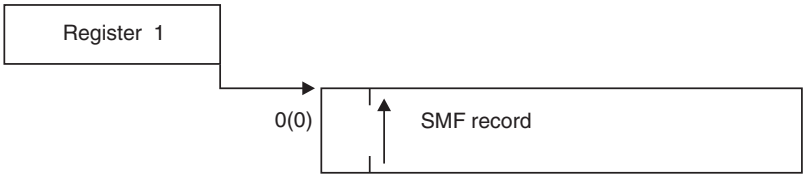

Figure 26. IEFU85 Input Parameter Structure

Return Specifications

A return code from IEFU85 indicates whether the current SMF record is to be suppressed.

If you associate multiple exit routines with IEFU85, you can specify how the return information is to be handled using the ATTRIB KEEPRC function of the SETPROG EXIT command, the EXIT statement of PROGxx, or CSVDYNEX services. If multiple exit routines match the ATTRIB KEEPRC criteria, the system returns information from the exit routine that finished first.

If you do not specify the ATTRIB KEEPRC function, the system returns the information from the exit routine whose return value was the greatest. If multiple exit routines return with the same highest value, the return information from the exit routine that finished first will be returned.

If you associate multiple exit routines with exit IEFU85, and any of those exit routines return with a value of 4, the current SMF record is suppressed.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-14 Same as at entry

15 One of the following return codes:

Return Code
Explanation

Value of 4
SMF is not to write the record to the SMF data set.

Other than 4
SMF is to write the record to the SMF data set.

Chapter 40. Global Resource Serialization Exits

Topics for The GRS Exits Appear as Follows:

System Programmer or Authorized Exits

ISGNQXITFAST — Fast ISGENQ / ENQ / DEQ Installation Exit

ISGNXITFAST is the IBM recommended replacement for ISGNQXIT and should be used except where restrictions apply.

For each ENQ/DEQ/RESERVE request with SCOPE=SYSTEM or SCOPE=SYSTEMS, the system invokes the Fast ENQ/DEQ Installation Exit, ISGNQXITFAST. The exit routine can modify attributes of the request prior to Resource Name List (RNL) processing. See *z/OS MVS Planning: Global Resource Serialization* for the installation flow through ENQ/DEQ exits.

By altering the ISGNXITFAST exit parameter list, the exit can:

- Alter the resource name (QNAME and/or RNAME).
- Alter the resource scope. See restrictions below .
- Alter the UCB address (for a RESERVE). See restrictions below .
- Convert a RESERVE to an ENQ by setting the UCB to zero. See restrictions below .
- Convert an ENQ to a RESERVE by adding a UCB specification. See restrictions below .
- Indicate to bypass the RNL processing.

Note: The exit routine cannot change the scope nor UCB address of a request when the program issues the ENQ or ISGENQ macro with RNL=NO. The Nqxp_SF1_RnlEqNo bit passed in the parameter list indicates if RNL=NO was specified.

This exit is invoked under the caller's unit of work on the system where the caller is running. For global resource requests, the exit is invoked only on the system where the request is made.

Note:

1. This exit is intended to replace ISGNQXIT. Although it is possible to run with both the ISGNQXIT and ISGNQXITFAST exits on a system, it is not recommended as path length and processing time are increased.
2. If both the ISGNQXIT and ISGNQXITFAST exits are installed, because ISGNQXIT is called second, any changes requested by the ISGNQXIT exits will override any changes made by the ISGNQXITFAST exits.
3. Any changes made by ISGNQXITFAST are not honored for an ISGENQ CHANGE or RELEASE request. Instead, the original changes made by ISGNQXITFAST on the OBTAIN are used.

Replacing the Exit Routine

Unlike RNL changes, GRS does not know how an exit alters the resource identity of a request. Therefore, to maintain data integrity, do not make an exit change that alters the resource identity of any outstanding or in-flight ENQ or DEQ requests. The resource identity consists of the QNAME, RNAME, SCOPE, and hardware reserve status. When you make exit changes, first stop the programs that are currently using the resource, and do not resume the programs until all the exit changes in the GRS complex have completed.

For information regarding dynamic exit routine replacement, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

ISGNQXITFAST receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In primary mode with H not= P not= S or H=P=S.
- In AMODE 31 and RMODE ANY.
- With no locks held or with the local and CMSEQDQ locks held.
- An FRR held when locks are held, otherwise an ARR is in effect.

Exit Recovery is as follows.

- If an error occurs, ISGNQXITFAST provides its own recovery routine.
- If no recovery exists, or the recovery continues with termination, GRS fails the request and continues processing. CSVDYNEX rules for the exit are used to determine whether the exit should be removed.

  The unplanned removal of an exit can result in serialization changes. Therefore, it is important to provide recovery and to understand how CSVDYNEX determines when to remove an abend exit. For more information about CSVDYNEX, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Exit Routine Processing

The ISGNQXITFAST exit routine is invoked for every ISGENQ/ENQ/DEQ/ RESERVE request issued for a resource. If any exit routines are defined to the dynamic exits facility, those routines are invoked before Resource Names List processing. If any ISGNQXIT exit routines are defined, these are run after the ISNGQXITFAST exit routines are run.

By updating and using information in the parameter list, the exit routine alters the following characteristics of the request:

- Resource major name (QNAME).
- Resource minor name (RNAME).
- Resource scope. If the requestor specified RNL=NO, changes to this parameter are not honored.
- Device UCB address (for RESERVE or DEQ with UCB requests). If the requestor specified RNL=NO, changes to this parameter are not honored. A UCB address can be deleted from a RESERVE request, converting the request from a RESERVE to an ENQ or a UCB address can be added to an ENQ, converting the request to a RESERVE.

The exit routine can also indicate that RNL processing can be bypassed.

Programming Considerations

Observe the following conventions when coding a Fast ISGENQ/ENQ/DEQ exit routine:

- Every exit routine must be reentrant.
- Because the exit is called for every ENQ, RESERVE, and DEQ request, an increased path length increases processor utilization and can degrade performance.
- Do not code the exit routine to issue the WAIT macro or call a service, such as WTOR, that issues a WAIT.
- Do not code the exit routine to issue another ISGENQ, ENQ, RESERVE, or DEQ macro.

Entry Specifications

The system passes a NQXP parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the exit parameter list (ISGYNQXP)
2-12	Not applicable
13	Address of a 72-byte save area
14	Return address
15	Entry point address of ISGNQXITFAST

Parameter List Contents: Register 1 contains the address of the exit parameter (NQXP) that is mapped by macro ISGYNQXP. Refer to *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the ISGYNQXP data area.

Return Specifications

The ISGNQXITFAST installation exit sets the appropriate request flag and alters the value in the parameter list. For example, to change the major name, the exit sets Nqxp_RFI_ChangeQName to B'1' and Nqxp_CP_QNAME to the new major name.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents on entry
15	0

ISGNQXIT — ISGENQ / ENQ / DEQ Installation Exit

For the best performance, IBM recommends that the installation use the ISGNQXITFAST rather than ISGNQXIT. The ISGNQXIT exit should only be used if the ISGNQXITFAST exit cannot be used. In no case should both exits be used because both will be called on every ENQ resulting in degraded system performance.

For each ISGENQ/ENQ/DEQ/RESERVE request with SCOPE=SYSTEM or SCOPE=SYSTEMS, the system invokes the ENQ/DEQ Installation Exit point, ISGNQXIT. The exit routines can modify attributes of the request prior to Resource Names List (RNL) processing. See *z/OS MVS Planning: Global Resource Serialization* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the installation flow through ENQ/DEQ exits.

By altering the exit parameter list, the exit can:

- Alter the resource name (QNAME and/or RNAME).
- Alter the resource scope. See restrictions below .
- Alter the UCB address (for a RESERVE). See restrictions below .
- Convert a RESERVE to an ENQ by setting the UCB to zero. See restrictions below .
- Convert an ENQ to a RESERVE by adding a UCB specification. See restrictions below .
- Indicate to bypass the RNL processing.

Note: The exit routine cannot change the scope nor UCB address of a request when the program issues the ENQ or ISGENQ macro with RNL=NO. The Nqxp_SF1_RnlEqNo bit passed in the parameter list indicates if RNL=NO was specified.

This exit is invoked under the caller's unit of work on the system where the caller is running. For global resource requests, the exit is invoked only on the system where the request is made.

Note:

1. Any changes made by ISGNQXIT are not honored for an ISGENQ CHANGE or RELEASE request. Instead, the original changes made by ISGNQXIT on the OBTAIN are used.
2. This exit replaces the ISGGREX0 exit interface, which was removed in z/OS V1R2; ISGGSIEX, ISGGSEEX, and ISGGRCEX are no longer used. Message ISG351I is issued and the exit is not invoked if any of these exits are installed.

Replacing the Exit Routine

Unlike RNL changes, GRS does not know how an exit alters the resource identity of a request. Therefore, to maintain data integrity, do not make an exit change that alters the resource identity of any outstanding or in-flight ENQ or DEQ requests. The resource identity consists of the QNAME, RNAME, SCOPE, and hardware reserve status. When you make exit changes, first stop the programs that are currently using the resource, and do not resume the programs until all the exit changes in the GRS complex have completed.

Any changes the exit makes must be taken into consideration. For information regarding dynamic exit routine replacement, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

ISGNQXIT receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In primary mode with H = P = S = requestor's address space.
- In AMODE 31 and RMODE ANY.
- With no locks held.

Exit Recovery is as follows.

- If an error occurs, ISGNQXIT provides its own recovery routine.
- If no recovery exists, or the recovery continues with termination, GRS fails the request and continues processing. CSVDYNEX rules for the exit are used to determine whether the exit should be removed.

  The unplanned removal of an exit can result in serialization changes. Such changes might cause data integrity errors. Therefore, it is important to provide recovery and to understand how CSVDYNEX determines when to remove an abend exit. For more information about CSVDYNEX, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Exit Routine Processing

The ISGNQXIT exit routine is invoked for every ENQ/DEQ/RESERVE/ISGENQ SCOPE=SYSTEM or SCOPE=SYSTEMS request issued for a resource. If any exit routines are defined to the dynamic exits facility, those routines are invoked before Resource Names List (RNL) processing.

By updating and using information in the parameter list, the exit routine alters the following characteristics of the request:

- Resource major name (QNAME).
- Resource minor name (RNAME).

- Resource scope. If the requester specified RNL=NO, changes to this parameter are not honored.
- Device UCB address (for RESERVE or DEQ with UCB requests). If the requester specified RNL=NO, changes to this parameter are not honored. A UCB address can be deleted from a RESERVE request, converting the request from a RESERVE to an ENQ or a UCB address can be added to an ENQ, converting the request to a RESERVE.

The exit routine can also indicate that RNL processing can be bypassed.

Programming Considerations

Observe the following conventions when coding an ISGENQ, ENQ, DEQ and RESERVE exit routine:

- Every exit routine must be reentrant.
- Because the exit is called for every ISGENQ, ENQ, DEQ, and RESERVE request, an increased path length increases processor utilization and can degrade performance.
- Do not code the exit routine to issue the WAIT macro or call a service, such as WTOR, that issues a WAIT.
- Do not code the exit routine to issue another ISGENQ, ENQ, RESERVE, or DEQ macro.
- The QNAME, RNAME, SCOPE, and UCB must match both the ENQ and the DEQ.
- Issuing the GETMAIN or STORAGE OBTAIN macro can slow down performance.

Entry Specifications

The ISGENQ/ENQ/DEQ/RESERVE mainline routine passes the address of the ENQ exit parameter list (ISGYNQXP).

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	**Contents**
0	Not applicable
1	Address of the exit parameter list (ISGYNQXP).
2-12	Not applicable
13	Address of a 72-byte save area
14	Return address
15	Entry point address of ISGNQXIT

Parameter List Contents: Register 1 contains the address of the exit parameter (NQXP) that is mapped by macro ISGYNQXP. Refer to *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the ISGYNQXP data area.

Return Specifications

The ISGENQ/ENQ/DEQ installation exit sets the appropriate request flag and alters the value in the parameter list. For example, to change the major name, the exit sets Nqxp_RFI_ChangeQName to '1'b and Nqxp_CP_QNAME to the new major name.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents on entry
15	0

ISGCNFXITSYSTEM — Filter Global Resource Serialization Contention Notification, SYSTEM Scope

ISGCNFXITSYSTEM provides a way for your installation to suppress the ENF 51 signal that global resource serialization issues to notify programs of contention for a scope=SYSTEM resource. When at least one exit routine has been added to the ISGCNFXITSYSTEM exit, it will receive control for every resource of system scope that Global Resource Serialization finds in contention. Depending on what your installation specified in the exit routine, the exit can suppress the ENF 51 signal for the resource in contention. This allows the installation to suppress ENF signals for resources known to be frequently in contention, with the trade-off of providing less data to any monitoring tools that listen for those signals.

Replacing the Exit Routine

For information regarding dynamic exit routine replacement, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

ISGCNFXITSYSTEM receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In primary ASC mode with H=S=requestor's address space, and P=Global Resource Serialization's address space.
- In task mode.
- In AMODE 31 and RMODE ANY.
- With no locks held, a local lock held, or both a local lock and the CMSEQDQ lock held..

Exit Recovery is as follows.

- If an error occurs, ISGCNFXITSYSTEM provides with its own recovery routine.
- If no recovery exists, or the recovery continues with termination, GRS fails the request and continues processing. CSVDYNEX rules for the exit are used to determine whether the exit should be removed.

  The unplanned removal of an exit can result in serialization changes. Such changes might cause data integrity errors. Therefore, it is important to provide recovery and to understand how CSVDYNEX determines when to remove an

abend exit. For more information about CSVDYNEX, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Exit Routine Processing

The ISGCNFXITSYSTEM exit point allows an exit routine to suppress the ENF 51 signal when global resource serialization detects contention on a system scope resource. The exit routine is passed a parameter list, CNFP (mapped by CNFP), which contains information about the request. The exit routine can check the QNAME, RNAME, and Rnamelen of the resource and filter the ENF 51 signal.

Programming Considerations

Observe the following considerations when coding an exit routine to filter global resource seriaqlization contention notification on a system scope:

- Every exit routine must be reentrant.
- While coding the exit routine, be aware that you will have an increased path length for ENQ/DEQ requests because the exit is called for every system scope resource in contention.
- Depending on the locks held upon entry to the exit routine, failing to return control promptly can cause significant delays in global resource serialization processing.
- Make sure that your exit routine takes into account the parameter list version, CNFP_VERSION, in case of future interface changes for the exit point.
- Do not code the exit routine to change any variables in the parameter list except for the filter flag. The exit does not make a copy of the resource identifying parameters passed, such as QNAME, RNAME, and Rnamelen.
- If you want to write an exit routine that covers both system (ISGCNFXITSYSTEM) and systems (ISGCNFXITSYSPLEX) scope resource contention, then use the CNFP_SYSTEM flag in the CNFP parameter list to differentiate between them.
- The resource described in the parameter list via CNFP_QNAME@, CNFP_RNAME@, CNFP_RLEN, and CNFP_SYSTEM reflects any changes made from other exits or RNL processing.

Entry Specifications

The system passes a parameter list (CNFP) to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the exit parameter list (ISGYCNFP)
2-12	Not applicable
13	Address of a 72-byte save area
14	Return address
15	Entry point of ISGCNFXITSYSTEM

Parameter List Contents: Register 1 contains the address of the exit parameter (CNFP) that is mapped by macro ISGYCNFP. The parameter list contains the

QNAME, RNAME, and Rnamelen of the resource, the filter flag, and a 4K work area. You should not use this work area to communicate among multiple exit routines because the work area may not be cleared upon entry. In addition, in certain recovery scenarios the work area may not be the same for all routines.

Return Specifications

At the completion of ISGCNFXITSYSTEM processing, ISGYCNFP can indicate that the ENF 51 signal be suppressed by setting the CNFP_FILTER flag.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents on entry
15	0

Coded Example of the Exit Routine

IBM provides coded examples of the ISGCNFXITSYSTEM and ISGCNFXITSYSPLEX exit routines in member ISGCNFXR of SYS1.SAMPLIB.

ISGCNFXITSYSPLEX — Filter Global Resource Serialization Contention Notification, SYSTEMS Scope

ISGCNFXITSYSPLEX provides a way for your installation to suppress the ENF 51 signal that global resource serialization issues to notify programs of contention for a scope=SYSTEMS resource. When at least one exit routine has been added to the ISGCNFXITSYSPLEX exit, it will receive control for every resource of systems scope that global resource serialization finds in contention. Depending on what your installation specified in the exit routine, the exit can suppress the ENF 51 signal for the resource in contention. This allows the installation to suppress ENF signals for resources known to be frequently in contention, with the trade-off of providing less data to any monitoring tools that listen for those signals.

Replacing the Exit Routine

For information regarding dynamic exit routine replacement, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

ISGCNFXITSYSPLEX receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In primary ASC mode with P=H=S=Global Resource Serialization's address space
- In task mode.
- In AMODE 31 and RMODE ANY.
- With no locks held, a local lock held, or both a local lock and the CMSEQDQ lock held..

Exit Recovery is as follows.

- If an error occurs, ISGCNFXITSYSPLEX provides with its own recovery routine.
- If no recovery exists, or the recovery continues with termination, GRS fails the request and continues processing. CSVDYNEX rules for the exit are used to determine whether the exit should be removed.

  The unplanned removal of an exit can result in serialization changes. Such changes might cause data integrity errors. Therefore, it is important to provide recovery and to understand how CSVDYNEX determines when to remove an abend exit. For more information about CSVDYNEX, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Exit Routine Processing

The ISGCNFXITSYSPLEX exit point allows an exit routine to suppress the ENF 51 signal when global resource serialization detects contention on a systems scope resource. The exit routine is passed a parameter list, CNFP (mapped by CNFP), which contains information about the request. The exit routine can check the QNAME, RNAME, and Rnamelen of the resource and filter the ENF 51 signal.

Programming Considerations

Observe the following considerations when coding an exit routine to filter global resource serialization contention notification on a systems scope:

- Every exit routine must be reentrant.
- While coding the exit routine, be aware that you will have an increased path length for ENQ/DEQ requests because the exit is called for every system scope resource in contention.
- Depending on the locks held upon entry to the exit routine, failing to return control promptly can cause significant delays in global resource serialization processing.
- Make sure that your exit routine takes into account the parameter list version, CNFP_VERSION, in case of future interface changes for the exit point.
- Do not code the exit routine to change any variables in the parameter list except for the filter flag. The exit does not make a copy of the resource identifying parameters passed, such as QNAME, RNAME, and Rnamelen.
- If you want to write an exit routine that covers both system (ISGCNFXITSYSTEM) and systems (ISGCNFXITSYSPLEX) scope resource contention, then use the CNFP_SYSTEM flag in the CNFP parameter list to differentiate between them.
- The resource described in the parameter list via CNFP_QNAME@, CNFP_RNAME@, CNFP_RLEN, and CNFP_SYSTEM reflects any changes made from other exits or RNL processing.

Entry Specifications

The system passes a parameter list (CNFP) to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	**Contents**
0	Not applicable
1	Address of the exit parameter list (ISGYCNFP)
2-12	Not applicable

13 Address of a 72-byte save area

14 Return address

15 Entry point of ISGCNFXITSYSPLEX

Parameter List Contents: Register 1 contains the address of the exit parameter (CNFP) that is mapped by macro ISGYCNFP. The parameter list contains the QNAME, RNAME, and Rnamelen of the resource, the filter flag, and a 4K work area. You should not use this work area to communicate among multiple exit routines because the work area may not be cleared upon entry. In addition, in certain recovery scenarios the work area may not be the same for all routines.

Return Specifications

At the completion of ISGCNFXITSYSPLEX processing, ISGYCNFP can indicate that the ENF 51 signal be suppressed by setting the CNFP_FILTER flag.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents on entry
15	0

Coded Example of the Exit Routine

IBM provides coded examples of the ISGCNFXITSYSPLEX and ISGCNFXITSYSTEM exit routines in member ISGCNFXR of SYS1.SAMPLIB.

Authorized Exits

ISGDGRSRES — Display Global Resource Serialization Resource Exit

ISGDGRSRES is called for each ENQ resource displayed in a DISPLAY GRS,RES= or CONTENTION command. It allows an application to provide additional information describing the meaning of the ENQ resource name.

The parameter list passed to ISGDGRSRES is mapped by ISGYDSPX (DSPX). The parameter list describes an ENQ resource name.

Replacing the Exit Routine

For information regarding dynamic exit routine replacement, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

ISGDGRSRES receives control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In primary mode with H = P = S = requestor's address space.
- In AMODE 31 and RMODE ANY.

- With no locks held.
- An EUT FRR is held when the exit routine is called.

Exit Recovery is as follows.

- If an error occurs, ISGDGRSRES provides with its own recovery routine.
- If no recovery exists, or the recovery continues with termination, GRS fails the request and continues processing. CSVDYNEX rules for the exit are used to determine whether the exit should be removed.

  The unplanned removal of an exit can result in serialization changes. Such changes might cause data integrity errors. Therefore, it is important to provide recovery and to understand how CSVDYNEX determines when to remove an abend exit. For more information about CSVDYNEX, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Exit Routine Processing

The exit routine should interrogate the QNAME and RNAME information that will be displayed. If the exit routine can add information about the meaning of the QNAME and RNAME for the resource, it can place:

- An identifier (16 EBCDIC characters padded with EBCDIC blanks on the right, if needed) in the Dspx_ResourceIdentifier field (for example, a subsystem or product name).
- Descriptive information (up to 70 EBCDIC characters padded with EBCDIC blanks on the right, if needed) in the Dspx_ResourceInformation field.

For an example, see Providing ENQ resource information on DISPLAY GRS in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Programming Considerations

Observe the following conventions when coding a display global resource serialization exit routine:

- Every exit routine must be reentrant.
- Be aware while coding the exit routine that because the exit is called for every resource displayed in a DISPLAY GRS command response, excessive path length could degrade system performance.
- Do not code the exit routine to issue the WAIT macro or call a service, such as WTOR, that issues a WAIT.
- Do not code the exit routine to issue another ENQ, RESERVE, or DEQ macro.

Entry Specifications

The system passes a ISGYDSPX parameter list (DSPX) to the exit routine. The DSPX contains the QNAME, RNAME, and scope of the resource being displayed. The exit routine can update the resource identifier and information fields for display.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the exit parameter list (ISGYDSPX)

2-12 Not applicable

13 Address of a 72-byte save area

14 Return address

15 Entry point address of ISGDGRSRES

Parameter List Contents: Register 1 contains the address of the exit parameter (DSPX) that is mapped by macro ISGYDSPX. Refer to *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the ISGYDSPX data area.

Return Specifications

The information displayed in the command output will be that from the last exit routine to update the DSPX parameter list.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents on entry
15	0

Authorized Exits for Alternate Serialization Products

ISGNQXITPREBATCH — ISGENQ / ENQ / DEQ Batch Preprocessing Exit

ISGNQXITPREBATCH provides the ability to reduce CPU consumption for ENQ, DEQ, ISGENQ or RESERVE requests that need to be processed by the ISGNQXITBATCH and ISGNQXITQUEUED1 exit points. The system calls the ISGNQXITPREBATCH exit point once for each ENQ, DEQ, ISGENQ or RESERVE resource with SCOPE=SYSTEM or SCOPE=SYSTEMS. If any of the ISGNQXITPREBATCH exit routines indicate that the resource should be processed by the ISGNQXITBATCHCND exit, then all of the resources in the current request will be presented to the ISGNQXITBATCHCND exit. Any ENQ or RESERVE request presented to the ISGNQXITBATCHCND exit will also be processed by the ISGNQXITQUEUED1 exit. See ENQ/DEQ Exits Installation in *z/OS MVS Planning: Global Resource Serialization* for the installation flow through ENQ/DEQ exits.

The ISGNQXITPREBATCH exit point also provides the ability to control future calls to this exit point. If the set of ISGNQXITPREBATCH exit routines indicates that this exit should no longer be called for a particular resource QNAME and scope, an entry for that QNAME is added to the GRS Exit Cache. Before calling the ISGNQXITPREBATCH exit point, GRS queries the GRS Exit Cache. If a matching entry is found and the entry indicates to NOT call the ISGNQXITPREBATCH exit point, this exit can be bypassed. Filtering the GRS Exit Cache is based on the QNAME, the original scope, the final scope, the original RESERVE state, and the final RESERVE state.

To restore processing of a resource through the ISGNQXITPREBATCH exit, the GRS Exit Cache must be cleared. For more information, see ISGGCECR, ClearCache call in *z/OS MVS Programming: Callable Services for High-Level Languages*.

This exit is invoked under the caller's unit of work on the system where the caller is running. For global resource requests, the exit is invoked only on the system where the request is made.

Note:

1. The ISGNQXITBATCH exit is not affected by the ISGNQXITPREBATCH exit.
2. After a call to ISGGCECR, the ISGNQXITPREBATCH exit can called again for all resources.
3. If an error occurs in the GRS Exit Cache, the cache will be disabled and all resources will again be processed by the ISGNQXITPREBATCH. You must do the IPL again to restore cache processing.

Note:

Replacing the Exit Routine

Unlike RNL changes, GRS does not know how an exit alters the resource identity of a request. Therefore, to maintain data integrity, do not make an exit change that alters the resource identity of any outstanding or in-flight ENQ or DEQ requests. The resource identity consists of the QNAME, RNAME, SCOPE, and hardware reserve status. When you make exit changes, first stop the programs that are currently using the resource, and do not resume the programs until all the exit changes in the GRS complex have completed.

For information regarding dynamic exit routine replacement, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

ISGNQXITPREBATCH receives control in the following environment:

- Task mode (running under the requester's task).
- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In primary ASC mode.
- In cross-memory mode with H=Caller's home address space, S=Any space, and P=Global Resource Serialization’s address space.
- In AMODE 31 and RMODE ANY.
- With no locks held or with the local and CMSEQDQ locks held.
- An FRR is held when locks are held, otherwise an ARR is in effect.

The routine can be invoked with an FRR established that must remain in effect. Therefore, system services or instructions that do not allow FRRs to be active or cause FRRs to be removed cannot be used. This includes the SVC instruction.

Exit Recovery is as follows.

- If an error occurs, the ISGNQXITPREBATCH exit routine should provide its own recovery routine.
- Exclusive use of EUT FRR recovery is the suggested recovery to be used because it can be used in all possible entry environments. ESTAE-like recovery is a worse performer and can be used only when locks are not held on entry. ESTAE-like recovery can be established but it does not receive control because an EUT FRR is established on entry.

- If no recovery exists, or the recovery continues with termination, GRS fails the request and continues processing. CSVDYNEX rules for the exit are used to determine whether the exit should be removed.

  The unplanned removal of an exit can result in serialization changes. Therefore, it is important to provide recovery and to understand how CSVDYNEX determines when to remove an abend exit. For more information about CSVDYNEX, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Exit Routine Processing

The ISGNQXITPREBATCH exit routine is invoked for every ENQ, DEQ, ISGENQ or RESERVE SCOPE=SYSTEM or SCOPE=SYSTEMS request issued for a resource. If any exit routines are defined to the dynamic exits facility, those routines are invoked after Resource Names List (RNL) processing.

By updating and using information in the parameter list, the exit routine indicates whether the ISGNQXITBATCHCND exit needs to be invoked or if the ISGNQXITPREBATCH exit will be called again for a resource.

Programming Considerations

Observe the following conventions when coding a Batch Preprocessing exit routine:

- Every exit routine must be reentrant.
- You should be aware while coding the exit routine, that because the exit is called for every ENQ, DEQ, ISGENQ and RESERVE request, an increased path length increases processor utilization and can degrade performance.
- Do not code the exit routine to issue the WAIT macro or call a service, such as WTOR, that issues a WAIT.
- Do not code the exit routine to issue another ENQ, DEQ, ISGENQ or RESERVE macro.

Entry Specifications

The system passes the NQPB parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the exit parameter list (NQPB)
2-12	Not applicable
13	Address of a 72-byte save area
14	Return address
15	Entry point address of ISGNQXITPREBATCH

Parameter List Contents: Register 1 contains the address of the exit parameter (NQPB) that is mapped by macro ISGYNQPB. Refer to *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the ISGYNQPB data area.

Return Specifications

The pre-batch installation exit sets the appropriate request flag in the parameter list to indicate:

- That ISGNQXITBATCHCND should be called
- That the scope of the ENQ should be considered to be global. This indication will be reported through ISGQUERY and ISGENQ TEST for API users to determine the overall scope of their ENQ request.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents on entry
15	0

ISGNQXITBATCH— ISGENQ / ENQ / DEQ Batched Exit ISGNQXITBATCHCND — ISGENQ / ENQ / DEQ Conditional Batch Processing Exit

Both the ISGNQXITBATCH exit and the ISGNQXITBATCHCND exit can interrogate and alter an entire ENQ, DEQ, ISGENQ, or RESERVE request. In most cases, the ISGNQXITBATCH exit and the ISGNQXITBATCHCND exit can be used interchangeably.

The ISGNQXITBATCH or ISGNQXITBATCHCND exit is invoked after all of the resources in the request have been passed to the ISGNQXIT exit point, and the RNLs have been processed for each resource in the request. Only SCOPE=SYSTEM and SCOPE=SYSTEMS resources are passed to the ISGNQXITBATCH or ISGNQXITBATCHCND exit; STEP requests are not passed to the ISGNQXITBATCH or ISGNQXITBATCHCND exit. See ENQ/DEQ Exits Installation in *z/OS MVS Planning: Global Resource Serialization* for the installation flow through ENQ/DEQ exits.

The ISGNQXITBATCH and ISGNQXITBATCHCND exits can:

- Set a return code for each resource in a conditional request. For example, make the request not happen.
- Convert a RESERVE to an ENQ.
- Set an ABEND for the request. For example, make the entire request not happen.

The parameter list passed to ISGNQXITBATCH (or ISGNQXITBATCHCND) exit is mapped by ISGYNQBP (NQBP). See ISGYNQBP macro in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for more information.

An ISGNQXITBATCH (or ISGNQXITBATCHCND) exit routine can set its own non-zero return code for each resource in the request, overriding ENQ or RESERVE functionality, for all:

- ENQ requests that specify a RET value or an ECB value
- Conditional ISGENQ requests

In addition, the ISGNQXITBATCH (or the ISGNQXITBATCHCND) exit routine can optionally set its own non-zero reason code that accompanies the non-zero return code for conditional ISGENQ requests.

For all requests (ENQ, DEQ, ISGENQ, and RESERVE), an ISGNQXITBATCH (or ISGNQXITBATCHCND) exit routine can set a one-byte ABEND code and a half-word reason code for the request. The one-byte ABEND code is used to generate the ABEND code. For example, if the exit routine sets the ABEND code to 5 for an ENQ request, a X'538' ABEND will be generated by ENQ processing.

When you use the ISGNQXITBATCH and ISGNQXITBATCHCND exits, be aware of the following notes.

Note:

1. ISGNQXITBATCH and ISGNQXITBATCHCND exits are only used to interrogate and alter an entire ENQ, DEQ, ISGENQ, or RESERVE request. For any other changes, you must use the ISGNQXIT exit or the ISGNQXITFAST exit.
2. If your installation is using an OEM serialization product to replace global resource serialization, do not specify that the exit is to be called last, either with the LAST parameter of the SETPROG command or the POS=LAST parameter of the CSVDYNEX macro, when installing the exit.
3. If all ENQ, DEQ, ISGENQ, and RESERVE requests need to be processed, use the ISGNQXITBATCH exit; if only a subset of requests need to be processed, use the combination of the ISGNQXITPREBATCH and ISGNQXITBATCHCND exit for better performance. See “ISGNQXITPREBATCH — ISGENQ / ENQ / DEQ Batch Preprocessing Exit” on page 273 for more information on the usage of ISGNQXITPREBATCH and ISGNQXITBATCHCND.
4. When both the ISGNQXITBATCH and ISGNQXITBATCHCND exits are installed, ISGNQXITBATCHCND is called second. Therefore, any actions requested by the ISGNQXITBATCHCND exits override any requests made by the ISGNQXITBATCH exits.

Replacing the Exit Routine

Unlike RNL changes, GRS does not know how an exit alters the resource identity of a request. Therefore, to maintain data integrity, do not make an exit change that alters the resource identity of any outstanding or in-flight ENQ or DEQ requests. The resource identity consists of the QNAME, RNAME, SCOPE, and hardware reserve status. When you make exit changes, first stop the programs that are currently using the resource, and do not resume the programs until all the exit changes in the GRS complex have completed.

For information regarding dynamic exit routine replacement, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

ISGNQXITBATCH (or ISGNQXITBATCHCND) exit receives control in the following environment:

- Task mode (running under the requester's task).
- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In primary ASC mode.

- In cross-memory mode with H=Caller's home address space, S=Any space, and P=Global Resource Serialization's address space.
- In AMODE 31 and RMODE ANY.
- With no locks held.
- An EUT FRR is held when the exit routine is called.

The routine can be invoked in the cross-memory mode with an EUT FRR established that must remain in effect. Therefore, system services or instructions that do not allow FRRs to be active or cause FRRs to be removed cannot be used. This includes the SVC instruction.

Exit Recovery is as follows.

- If an error occurs, the ISGNQXITBATCH (or ISGNQXITBATCHCND) exit should provide its own EUT FRR recovery routine.
- An EUT FRR recovery is the only recovery type that can be used. ESTAE-like recovery can be established but it does not receive control because an EUT FRR is established on entry.
- If no recovery exists, or the recovery continues with termination, GRS fails the request and continues processing. CSVDYNEX rules for the exit are used to determine whether the exit should be removed.

  The unplanned removal of an exit can result in serialization changes. Therefore, it is important to provide recovery and to understand how CSVDYNEX determines when to remove an abend exit. For more information about CSVDYNEX, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Exit Routine Processing

The ISGNQXITBATCH (or ISGNQXITBATCHCND) exit point allows an exit routine to interrogate an ENQ, DEQ, ISGENQ, or RESERVE request as a single unit, rather that on an individual resource basis, as ISGNQXIT. The exit routine is passed a NQBP (mapped by ISGYNQBP) which contains information about the request.

The exit routine can affect processing in the following ways:

1. Set an ABEND code for the entire request. The requestor is ABENDed with this ABEND code, prepended to the appropriate SVC number. For example, if the exit indicated to use ABEND code 5 on an ENQ request, the caller would get ABENDed with a S538 ABEND code.
2. Set the return code for a conditional request. If the request specified RET=TEST, RET=HAVE, USE, CHNG, or ISGENQ COND=YES the exit routine can prevent normal ENQ/DEQ processing by indicating the return code to pass back to the caller. If this return code is set, the system will do no further processing on that resource. The list of acceptable return codes is as follows:

   ENQ/RESERVE:

   - RET=TEST RC=X'00',X'04',X'08',X'14',X'20'
   - RET=USE RC=X'00',X'04',X'08',X'14',X'18'
   - RET=CHNG RC=X'00',X'04',X'08',X'14'
   - RET=HAVE RC=X'00',X'08',X'14',X'18'X'28',X'44'
   - RET=ECB RC=X'00',X'08',X'14',X'18'X'28',X'44'

   DEQ:

   - RET=HAVE RC=X'00',X'04',X'08'

ISGENQ:

- RC=4 RSN=0402,0404,0405,0406,0407,0409,040A,040B
- RC=8 RSN=0815
- RC=C RSN=0C05
- RC=10 RSN=FFyy where yy are diagnostic bits set by the exit

Failure to use an acceptable return code may result in unexpected results for the request. Example of failure case:

On an ENQ RET=HAVE, when an exit sets a resource return code to X'20' GRS will not process the request. Instead GRS will pass the X'20' back to the ENQ caller. The ENQ caller will assume they have control of the resource because the return code X'20' indicates so. But GRS did not process the request so when the corresponding DEQ is issued for the resource, GRS will not find the resource on its queues, and therefore issue an ABEND130.

3. If the ISGNQXITBATCH (or ISGNQXITBATCHCND) exit is driven by an ISGENQ COND=YES request, and ISGNQXITBATCH (or ISGNQXITBATCHCND) specifies a return AND reason code, only the lower two bytes of the reason code will be set; the upper two bytes are reserved for system use.
4. Convert a RESERVE to an ENQ.

Programming Considerations

Observe the following conventions when you code a Batch ENQ exit routine. If you do not want to get control for all request instances but only for particular requests, the combined use of the ISGNQXITPREBATCH and ISGNQXITBATCHCND provides better performance over the ISGNQXITBATCH exit. See “ISGNQXITPREBATCH — ISGENQ / ENQ / DEQ Batch Preprocessing Exit” on page 273 for more information.

- Every exit routine must be reentrant.
- Because the exit is called for every ENQ, DEQ, ISGENQ, and RESERVE request, an increased path length increases processor utilization and can degrade performance.
- Do not code the exit routine to issue the WAIT macro or call a service, such as WTOR, that issues a WAIT.
- Do not code the exit routine to issue another ENQ, DEQ, ISGENQ, or RESERVE macro.

Entry Specifications

The system passes a Batch Exit parameter list (NQBP) to the exit routine. The NQBP contains a header section describing the request and an entry (NQBPRSC) for each resource in the request.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the exit parameter list (ISGYNQBP)
2-12	Not applicable
13	Address of a 72-byte save area

14 Return address

15 Entry point address of ISGNQXITBATCH (or ISGNQXITBATCHCND)

Parameter List Contents: Register 1 contains the address of the exit parameter (NQBP) that is mapped by macro ISGYNQBP. Refer to *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the ISGYNQBP data area.

Return Specifications

At the completion of ISGNQXITBATCH (or ISGNQXITBATCHCND) processing, ISGYNQBP can indicate that an ABEND code or return code has been set, or that a RESERVE has been converted to an ENQ.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents on entry
15	0

ISGNQXITQUEUED1 — ISGENQ / ENQ / DEQ First Queued Exit

ISGNQXITQUEUED1 is called when all the elements of an ENQ (not DEQ) request have completed local processing or have been queued to the global processor. The exit is called just prior to waiting (for unconditional requests that have not been granted) or returning to the ENQ requester. ISGNQXITQUEUED1 is called for both successful and unsuccessful cases. However it is not called at all if neither the batch nor batch conditional exits were called. See ENQ/DEQ Exits Installation in *z/OS MVS Planning: Global Resource Serialization* for the installation flow through ENQ/DEQ exits.

The parameter list passed to ISGNQXITQUEUED1 is mapped by ISGYNQQP (NQQP). The request data presented to ISGNQXITQUEUED1 are read only. Information contained in NQQP includes return codes that have been set for the local resources, the ABEND code if the request failed, and whether the request needs to be suspended for RNL processing.

Note:

1. This exit is intended for use by OEM serialization products.
2. If your installation is using an OEM serialization product to replace global resource serialization, you should NOT specify that any installation provided exits are to be called first (either with the FIRST parameter of the SETPROG command or the POS=FIRST parameter of the CSVDYNEX macro) when installing this exit.

Replacing the Exit Routine

Unlike RNL changes, GRS does not know how an exit alters the resource identity of a request. Therefore, to maintain data integrity, do not make an exit change that alters the resource identity of any outstanding or in-flight ENQ or DEQ requests. The resource identity consists of the QNAME, RNAME, SCOPE, and hardware reserve status. When you make exit changes, first stop the programs that are

currently using the resource, and do not resume the programs until all the exit changes in the GRS complex have completed.

For information regarding dynamic exit routine replacement, see "Replacing a Dynamic Exit Routine" on page 6.

Exit Routine Environment

ISGNQXITQUEUED1 receives control in the following environment:

- Task mode (running under the requester's task).
- Enabled for interrupts.
- Supervisor state with PSW key 0.
- Cross-memory mode with H=S=Requester's address space and P=Global Resource Serialization's address space.
- AMODE 31 and primary ASC mode.
- With no locks held.
- An EUT FRR might be held when the exit routine is called.

The routine can be invoked in the cross-memory mode with an EUT FRR established that must remain in effect. Therefore, system services or instructions that do not allow FRRs to be active or cause FRRs to be removed cannot be used. This includes the SVC instruction.

Exit Recovery is as follows.

- If an error occurs, the ISGNQXITQUEUED1 exit should provide its own EUT FRR recovery routine.
- An EUT FRR recovery is the only recovery type that can be used. ESTAE-like recovery can be established but it does not receive control because an EUT FRR is established on entry.
- If no recovery exists, or the recovery continues with termination, GRS fails the request and continues processing. CSVDYNEX rules for the exit are used to determine whether the exit should be removed.

  The unplanned removal of an exit can result in serialization changes. Therefore, it is important to provide recovery and to understand how CSVDYNEX determines when to remove an abend exit. For more information about CSVDYNEX, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Exit Routine Processing

The ISGNQXITQUEUED1 exit point is called after the local resources in an ENQ/RESERVE rquest have been completed and the global resources have been queued to the global resource serialization address space for global processing. DEQ requests are not passed to ISGNQXITQUEUED1 exit routines. The exit routine is passed a NQQP (mapped by ISGYNQQP) which contains information about the request. If the request is going to ABEND, the ABEND code is indicated. An indicator is set if the request will be redriven due to a dynamic RNL change processing. Only SCOPE=SYSTEM and SYSTEMS resources are passed to the exit (STEP requests are not passed to the exit).

The NQQP is followed by one NQQPRSC entry for each resource in the request. Each NQQPRSC entry indicates the result for each local ENQ resource, or the fact that the request has been queued, for each global resource.

Programming Considerations

Most of the exits that are driven for a request are provided with a unique "request token". This token allows the exits to correlate any required user information between exit callers. For example, the Nqqp_RD_RequestToken will be the same as the Nqxp_RequestToken if both the ISGNQXITQUEUED1 and ISGNQXITFAST exits are driven for the same request.

Observe the following conventions when coding an ENQ/DEQ First Queued exit routine:

- Every exit routine must be reentrant.
- You should be aware while coding the exit routine, that because the exit is called for every ENQ and RESERVE request, an increased path length increases processor utilization and can degrade performance.
- Do not code the exit routine to issue the WAIT macro or call a service, such as WTOR, that issues a WAIT.
- Do not code the exit routine to issue another ENQ, RESERVE, or DEQ macro.
- The ISGNQXITQUEUED1 exit point is called for all ENQ/RESERVE requests only when either the ISGNQXITBATCH or ISGNQXITBATCHCND exits are called. In other words, only if a batch exit is called, the queued exit is called.

Entry Specifications

The system passes a NQQP parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the exit parameter list (ISGYNQQP)
2-12	Not applicable
13	Address of a 72-byte save area
14	Return address
15	Entry point address of ISGNQXITQUEUED1

Parameter List Contents: Register 1 contains the address of the exit parameter (NQQP) that is mapped by macro ISGYNQQP. Refer to *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the ISGYNQQP data area.

Return Specifications

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents on entry
15	0

ISGNQXITQUEUED2 – ISGENQ / ENQ / DEQ Second Queued Exit

ISGNQXITQUEUED2 is called when all the elements of an ENQ (not DEQ) request have completed both local and global processing. The exit is called before waiting (for unconditional requests that have not been granted) or returning to the ENQ requester. This exit is called only if either the ISGNQXITBATCH or ISGNQXITBATCHCND exits have been called previously for the same request.

See ENQ/DEQ Exits Installation in *z/OS MVS Planning: Global Resource Serialization* for the installation flow through ENQ/DEQ exits.

ISGYNQQP (NQQP) maps the parameter list that is passed to ISGNQXITQUEUED2. The request data presented to ISGNQXITQUEUED2 are read only. The following information is contained in NQQP:

- Return codes that have been set for both local and global resources.
- The ABEND code if the request failed.
- Information about whether the request needs to be suspended for RNL processing.

Note:

1. This exit is intended for use by OEM serialization products.
2. If your installation is using an OEM serialization product to replace global resource serialization, do not specify that any installation-provided exits are to be called first (either with the FIRST parameter of the SETPROG command or with the POS=FIRST parameter of the CSVDYNEX macro) when installing this exit.

Replacing the Exit Routine

For information about the replacement of the dynamic exit routine, see “Replacing a Dynamic Exit Routine” on page 6.

Exit Routine Environment

ISGNQXITQUEUED2 receives control in the following environment:

- Task mode (running under the requester’s task).
- Enabled for interrupts.
- Supervisor state with PSW key 0.
- Cross-memory mode with H=S=Requester’s address space and P=Global Resource Serialization’s address space.
- AMODE 31 and primary ASC mode.
- With no locks held.
- An EUT FRR might be held when the exit routine is called.

The routine can be invoked in the cross-memory mode with an EUT FRR established that must remain in effect. Therefore, system services or instructions that do not allow FRRs to be active or that cause FRRs to be removed cannot be used. This includes the SVC instruction.

Exit Recovery is as follows.

- To recover from errors, the ISGNQXITQUEUED2 routine must provide its own recovery routine.

- If no recovery exists, or the recovery terminates, GRS fails the request and continues processing. CSVDYNEX rules for the exit are used to determine whether the exit needs to be removed.

An unplanned removal of an exit can result in loss of the function it provides. Ensure that you provide recovery and understand how CSVDYNEX determines when to remove an exit that ends abnormally. For more information about CSVDYNEX, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Exit Routine Processing

The ISGNQXITQUEUED2 exit point is called after all local and global resources in an ENQ/RESERVE request have been processed. DEQ requests are not passed to ISGNQXITQUEUED2 exit routines. The exit routine is passed by an NQQP (mapped by ISGYNQQP) that contains information about the request. If the request ends abnormally, an abend code is indicated. Only SCOPE=SYSTEM and SYSTEMS resources are passed to the exit (STEP requests are not passed to the exit).

The NQQP is followed by one NQQPRSC entry for each resource in the request. Each NQQPRSC entry indicates the result for each local and global ENQ resource.

Programming Considerations

Most of the exits that are driven for a request are provided with a unique request token. This token enables the exits to correlate any required user information between exit callers. For example, the Nqqp_RD_RequestToken is the same as the Nqxp_RequestToken if both the ISGNQXITQUEUED2 and ISGNQXITFAST exits are driven for the same request.

Observe the following conventions when coding an ISGNQXITQUEUED2 exit routine:

- Every exit routine must be re-entrant.
- When you are coding the exit routine, an increased path length increases processor utilization and can degrade performance, because the exit is called for every ENQ, RESERVE and ISGENQ OBTAIN/CHANGE request.
- Do not code the exit routine to issue another ENQ, RESERVE, DEQ or ISGENQ macro or to call another service that might issue those macros.
- The ISGNQXITQUEUED2 exit point is called for all ENQ/RESERVE requests only when either the ISGNQXITBATCH or ISGNQXITBATCHCND exits are called. In other words, only if a batch exit is called, the queued exit is called.

Entry Specifications

The system passes a NQQP parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the exit parameter list (ISGYNQQP)
2-12	Not applicable

13 Address of a 72-byte save area

14 Return address

15 Entry point address of ISGNQXITQUEUED2

The contents of the registers on entry to ISGNQXITQUEUED2 are as follows: Access Register contents are not predictable.

Parameter List Contents: Register 1 contains the address of the exit parameter (NQQP) that is mapped by macro ISGYNQQP. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the ISGYNQQP data area.

Return Specifications

No return information is expected.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents on entry
15	Zero

ISGENDOFLQCB — End of Local QCB Exit

ISGENDOFLQCB is called when the last requester for a LOCAL resource (SCOPE=SYSTEM or RNL excluded SCOPE=SYSTEMS, but not SCOPE=STEP) is DEQed. The ISGENDOFLQCB exit is also called for SCOPE=SYSTEMS resources in a GRS=NONE environment. See ENQ/DEQ Exits Installation in *z/OS MVS Planning: Global Resource Serialization* for the installation flow through ENQ/DEQ exits.

The ISGENDOFLQCB exit point provides the ability to control future calls to this exit point. If the set of ISGENDOFLQCB exit routines indicates that this exit no longer needs to be called for a particular resource QNAME, an entry for that QNAME is added to the GRS Exit Cache. Prior to calling the ISGENDOFLQCB exit point, GRS queries the GRS Exit Cache. If a matching entry is found, and this entry indicates to NOT call the ISGENDOFLQCB exit point, then this exit may be bypassed. To restore processing of a resource through the ISGENDOFLQCB exit, the GRS Exit Cache must be cleared. See, ISGGCECR, ClearCache call in *z/OS MVS Programming: Callable Services for High-Level Languages*.

The parameter list passed to ISGENDOFLQCB is mapped by ISGYQCBP (QCBP). The parameter list describes a resource for which the last requester on this system has been DEQed.

Note:

1. This exit is intended for use by OEM serialization products.
2. If your installation is using an OEM serialization product to replace Global Resource Serialization, you should NOT specify that the exit is to be called last (either with the LAST parameter of the SETPROG command or the POS=LAST parameter of the CSVDYNEX macro) when installing this exit.

3. After a call to ISGGCECR, the ISGENDOFLQCB exit can be called again for all resources.
4. If an error occurs in the GRS Exit Cache, the cache is disabled and all resources will again be processed by the ISGENDOFLQCB exit. Re-IPL to restore cache processing.

Replacing the Exit Routine

Unlike RNL changes, GRS does not know how an exit alters the resource identity of a request. Therefore, to maintain data integrity, do not make an exit change that alters the resource identity of any outstanding or in-flight ENQ or DEQ requests. The resource identity consists of the QNAME, RNAME, SCOPE, and hardware reserve status. When you make exit changes, first stop the programs that are currently using the resource, and do not resume the programs until all the exit changes in the GRS complex have completed.

For information regarding dynamic exit routine replacement, see "Replacing a Dynamic Exit Routine" on page 6.

Exit Routine Environment

ISGENDOFLQCB receives control in the following environment:

- Task mode.
- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In primary ASC mode.
- Might be invoked in the cross-memory mode.
- In AMODE 31 and RMODE ANY.
- No locks held or local and CMSEQDQ locks held.
- An EUT FRR might be held when the exit routine is called.

The routine can be invoked in the cross-memory mode with an EUT FRR established that must remain in effect. Therefore, system services or instructions that do not allow FRRs to be active or cause FRRs to be removed cannot be used. This includes the SVC instruction.

Exit Recovery is as follows.

- If an error occurs, the ISGENDOFLQCB exit should provide its own EUT FRR recovery routine.
- An EUT FRR recovery is the only recovery type that can be used. ESTAE-like recovery can be established, but it does not receive control because an EUT FRR might be established on entry.
- If no recovery exists, or the recovery continues with termination, GRS fails the request and continues processing. CSVDYNEX rules for the exit are used to determine whether the exit should be removed.

  The unplanned removal of an exit can result in serialization changes. Therefore, it is important to provide recovery and to understand how CSVDYNEX determines when to remove an abend exit. For more information about CSVDYNEX, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Exit Routine Processing

The ISGENDOFLQCB exit point is called after the last requester for a local resource (SCOPE=SYSTEM, only) DEQs from that resource. The exit routine is passed a QCBP (mapped by ISGYQCBP) which describes the resource that is being deleted.

Programming Considerations

Observe the following conventions when coding an End of Local QCB exit routine:

- Every exit routine must be reentrant.
- You should be aware while coding the exit routine, that because the exit is called for every ENQ, RESERVE, and DEQ request, an increased path length increases processor utilization and can degrade performance.
- Do not code the exit routine to issue the WAIT macro or call a service, such as WTOR, that issues a WAIT.
- Do not code the exit routine to issue another ENQ, RESERVE, or DEQ macro.
- Do not code the exit such that the Local or CMSEQDQ locks are freed.

Entry Specifications

The system passes an QCBP parameter list to the exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Address of the exit parameter list (ISGYQCBP)
2-12	Not applicable
13	Address of a 72-byte save area
14	Return address
15	Entry point address of ISGENDOFLQCB

Parameter List Contents: Register 1 contains the address of the exit parameter (QCBP) that is mapped by macro ISGYQCBP. Refer to *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the mapping of the ISGYQCBP data area.

Return Specifications

The ISGENDOFLQCB installation exit sets the appropriate request flag in the parameter list to indicate if ISGENDOFLQCB exit routine will be called again.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register	Contents
0-14	Restored to contents on entry
15	0

Chapter 41. IXC_ELEM_RESTART — Element Restart Exit

Topics for This Exit Appear as Follows:

Through the IXC_ELEM_RESTART exit, your installation can modify or cancel the automatic restart management-initiated restart of an element. Your installation may use this exit to coordinate the restart of an element with other automation routines, and to make decisions about how, or if, it will be restarted. Automatic restart management (ARM) invokes this exit once for each element that is to be restarted, on the system where it will be restarted.

Installing the Exit Routine

IXC_ELEM_RESTART must be a load module in LPA or in an APF-authorized library in the LNKLST concatenation on all the systems in the sysplex that are, or may be, connected to the ARM couple data set. This exit must be linkedited as reentrant.

For general instructions on installing a dynamic exit routine, see "Link editing a Dynamic Exit Routine into a Library" on page 6.

Exit Routine Environment

The element restart exit receives control on the system where the element is to be restarted, in the following environment:

- Task mode
- Supervisor state with PSW key 0
- 31-bit addressing mode
- Primary ASC mode
- Enabled for interrupts
- No locks held

Exit Recovery: Your installation must provide recovery for this exit, if it is critical for the exit to successfully complete its processing. If you do not provide recovery and the exit ends abnormally, MVS restarts proceed without modification.

Exit Routine Processing

MVS invokes the IXC_ELEM_RESTART exit on the system on which the element is to be restarted, just before restarting the element. When MVS is restarting a group of elements from a failing system in the sysplex, MVS first invokes the IXC_WORK_RESTART exit once, so the installation can prepare the new system for additional work, and then invokes the IXC_ELEM_RESTART exit once **for each element** to be restarted.

When the exit routine receives control, MVS passes a parameter list (mapped by the IXCYERE macro) that contains such information as the name of the element to be restarted, the reason for the restart, and the restart method. Based on your knowledge of the installation, the systems in its sysplex, its automation packages and production controllers, and the elements registered with the automatic restart manager, you can design the element restart exit to use the parameter list information to do one of the following:

- Allow MVS to proceed without changes by setting a return code of 0.
- Select a different restart method; for example, request that MVS use a START command instead of persistent JCL.
  1. Check to see what restart methods are available in the EREFLAGS field.
  2. Update the EREJCLDATASET with the name of the data set that contains the JCL that should be used, or update ERESTARTTXT with the new START text that should be used.
  3. Indicate the change in the ERERESTARTTYPE field.
  4. Set a return code of 4.
- Modify the current restart method.
  1. Modify the START command text in ERESTARTTXT, or the JCL in EREJCLDATASET, whichever is appropriate.
  2. Set a return code of 4.
- Cancel the restart of the element.
  1. Indicate in the ERERESTARTTYPE field that the element should not be restarted.
  2. Set a return code of 4.

Programming Considerations

- Code IXC_ELEM_RESTART to be reentrant.
- If you provide new restart text (either a JCL data set name or a started task command), and you use any symbolic substitution parameters (such as &SYSCLONE.), they will be resolved using the values from the system on which the element initially registered. If you use symbolic substitution parameters, retrieve the replication ID of the system the element initially registered on, the system the element was previously running on, or the system the element is to be restarted on from the EREHOMESYSTEM field, the EREFROMSYSTEM field, or the ERETOSYSTEM field in IXCYERE.
- Do not code the IXC_ELEM_RESTART exit to perform the restart itself, because the element might not be able to reregister. Use this exit to notify automatic restart management that it should not restart the element. Use a separate program to restart the element, after it has been deregistered. To indicate that this element should not be restarted by automatic restart management, code the following in the IXC_ELEM_RESTART exit:
  – Set ERERESTARTTYPE to X'01' to indicate that automatic restart management should not restart the element.

– Set a return code of X'4' in register 15.

From another program, use the ENFREQ macro to listen for the deregister ENF signal for the element. Restart the element.

Entry Specifications

MVS passes to IXC_ELEM_RESTART the address of the parameter list mapped by IXCYERE.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Does not contain any information for use by the exit
1	Address of the IXCYERE parameter list
2-12	Does not contain any information for use by the exit
13	Register save area
14	Return address
15	Entry point address of IXC_ELEM_RESTART

Parameter List Contents: Register 1 contains the address of the element restart exit parameter list, which is mapped by the IXCYERE mapping macro. The parameter list contains:

- The job name, element name, and element type.
- An indication of whether the element registered with the ELEMBIND=CURSYS option.
- START command text, persistent JCL, or the data set name containing the JCL that MVS will use to restart the element.
- An indication of how MVS will restart the element (START command, persistent JCL, or JCL in a data set).
- The name of the system where the element will be restarted.
- The name of the system the element was running on when the failure occurred.
- An indication of whether persistent restart text is available or is not available for this element. Persistent restart text is either the JCL or the started task command that was used to previously start this element.
- An indication of whether the exit must supply restart text or the restart will fail and the element will be deregistered.

The IXCYERE mapping is described in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

MVS expects the exit to return a return code of 0 if no changes to the restart method were requested, or a return code of 4 if a change to the restart method was requested in the ERERESTARTTYPE field of IXCYERE. ERERESTARTTYPE may also be used to indicate that the automatic restart manager should not restart an element.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-14 The exit does not have to place any information in these registers, and does not have to restore their contents to what they were when the exit received control.

15 Return code

- **0** No changes to the restart method
- **4** The automatic restart manager should change the restart method as indicated in the IXCYERE parameter list.

Coded Example of the Exit Routine

The following is an example of an IXC_ELEM_RESTART exit.

```
TITLE 'ELEMEXIT - SAMPLE AUTOMATIC RESTART MANAGER ELEMENT RESTART EXIT'
***START OF SPECIFICATIONS*********************************************
*                                                                     *
* MODULE NAME         =  ELEMEXIT                                     *
*                                                                     *
* DESCRIPTIVE NAME    =  SAMPLE AUTOMATIC RESTART MANAGER ELEMENT     *
*                        RESTART EXIT.                                *
*                                                                     *
* FUNCTION            =  THIS EXIT WILL PROHIBIT RESTARTS FOR ELEMENT *
*                        TERMINATIONS BUT WILL ALLOW RESTARTS FOR     *
*                        SYSTEM TERMINATIONS.                         *
*                                                                     *
*   OPERATION         =  DETERMINES THE TERMINATION TYPE FROM THE     *
*                        EVENT CODE IN THE ELEMENT RESTART EXIT       *
*                        PARAMETER LIST. IF AN ELEMENT TERMINATION IS *
*                        INDICATED, THE RESTART TYPE IN THE PARAMETER *
*                        LIST WILL BE SET TO RESTART OF NONE AND THE  *
*                        EXIT'S RETURN CODE WILL BE SET TO FOUR.      *
*                        IF A SYSTEM TERMINATION IS INDICATED, THE    *
*                        PARAMETER LIST WILL NOT BE CHANGED AND THE   *
*                        EXIT'S RETURN CODE WILL BE SET TO ZERO.      *
*                                                                     *
* ENTRY POINT         =  ELEMEXIT                                     *
*                                                                     *
*   PURPOSE           =  TO DETERMINE THE WAY IN WHICH AN ELEMENT     *
*                        WILL BE RESTARTED.                           *
*                                                                     *
*   LINKAGE           =  BALR                                         *
*                                                                     *
*   INPUT DATA        =  REG1  ADDRESS OF THE IXCYERE PARAMETER LIST  *
*                        REG13 ADDRESS OF STANDARD SAVE AREA          *
*                        REG14 RETURN ADDRESS                         *
*                        REG15 ENTRY POINT ADDRESS                    *
*                                                                     *
*   REGISTERS SAVED   =  REG0 - REG15                                 *
*                                                                     *
*   REGISTER USAGE    =  REG0  - USED FOR BASING                      *
*                        REG1  - PARAMETER REGISTER                   *
*                        REG2  - NOT USED                             *
*                        REG3  - WORK REGISTER                        *
*                        REG4  - NOT USED                             *
*                        REG5  - POINTER TO IXCYERE                   *
*                        REG6  - NOT USED                             *
*                        REG7  - NOT USED                             *
*                        REG8  - NOT USED                             *
*                        REG9  - NOT USED                             *
*                        REG10 - NOT USED                             *
*                        REG11 - NOT USED                             *
```

```
*                       REG12 - MODULE BASE REGISTER                *
*                       REG13 - POINTER TO A STANDARD SAVE AREA     *
*                       REG14 - RETURN POINT                        *
*                       REG15 - RETURN CODE                         *
*                                                                   *
*   REGISTERS RESTORED = REG0 - REG14                               *
*                                                                   *
*   CONTROL BLOCKS    =                                             *
*     NAME     MAPPING MACRO   REASON USED                 USAGE    *
*     ----     -------------   -----------                -------   *
*     ERE       IXCYERE         EXIT PARAMETER LIST         R,W     *
*                                                                   *
*   KEY = R-READ, W-WRITE, C-CREATE, D-DELETE                       *
*                                                                   *
*   TABLES            =  NONE                                       *
*                                                                   *
*   MACROS            =  NONE                                       *
*                                                                   *
*   MESSAGES          =  NONE                                       *
*                                                                   *
*   MODULE TYPE       = CSECT                                       *
*                                                                   *
*     ATTRIBUTES      =  REENTRANT, REUSABLE, AMODE 31, RMODE ANY   *
*                                                                   *
*********************************************************************
         EJECT
ELEMEXIT CSECT
ELEMEXIT AMODE 31                    31-BIT ADDRESSING MODE
ELEMEXIT RMODE ANY                   31-BIT RESIDENCE
         SPACE 1
*********************************************************************
*                                                                   *
*         REGISTER ASSIGNMENTS                                      *
*                                                                   *
*********************************************************************
REG0     EQU   0                      REGISTER 0
REG1     EQU   1                      REGISTER 1
REG2     EQU   2                      REGISTER 2
REG3     EQU   3                      REGISTER 3
REG4     EQU   4                      REGISTER 4
EREPTR   EQU   5                      REGISTER 5 - POINTS TO ERE
REG6     EQU   6                      REGISTER 6
REG7     EQU   7                      REGISTER 7
REG8     EQU   8                      REGISTER 8
REG9     EQU   9                      REGISTER 9
REG10    EQU   10                     REGISTER 10
REG11    EQU   11                     REGISTER 11
REG12    EQU   12                     REGISTER 12
BASEREG  EQU   12                     REGISTER 12 - MODULE BASE
REG13    EQU   13                     REGISTER 13
REG14    EQU   14                     REGISTER 14
REG15    EQU   15                     REGISTER 15
         EJECT
*********************************************************************
*                                                                   *
*        STANDARD ENTRY LINKAGE                                     *
*                                                                   *
*********************************************************************
         STM   REG14,REG12,12(REG13)    SAVE CALLER'S REGISTERS
         BALR  BASEREG,REG0             ESTABLISH MODULE BASE
         USING *,BASEREG                ESTABLISH ADDRESSABILITY
         LR    EREPTR,REG1              ESTABLISH ADDRESSABILITY
         USING ERE,EREPTR               TO THE ERE
*********************************************************************
*                                                                   *
*        DETERMINE THE TERMINATION TYPE BASED ON THE INPUT          *
*        EVENT CODE.                                                *
```

```
*                                                                     *
***********************************************************************
         SPACE 1
         LA    REG3,EREELEMTERM
         CH    REG3,EREEVENTCODE        ELEMENT TERMINATION?
         BNE   SYSTERM                  NO, GO TO SYSTEM TERMINATION
***********************************************************************
*                                                                     *
*        IF ELEMENT TERMINATION IS INDICATED, THE RESTART TYPE IN     *
*        THE PARAMETER LIST IS SET TO RESTART OF NONE AND THE         *
*        EXIT'S RETURN CODE IS SET TO FOUR.                           *
*                                                                     *
***********************************************************************
         MVI   ERERESTARTTYPE,ERERESTARTNONE  SET RESTART TO NONE
         L     REG15,FOUR               SET RETURN CODE TO FOUR
         B     FINISHED                 GO TO FINISHED
***********************************************************************
*                                                                     *
*        IF SYSTEM TERMINATION IS INDICATED, THE PARAMETER LIST       *
*        IS NOT CHANGED AND THE EXIT'S RETURN CODE IS SET TO ZERO.    *
*                                                                     *
***********************************************************************
SYSTERM  EQU   *
         L     REG15,ZERO               SET RETURN CODE TO ZERO
***********************************************************************
*                                                                     *
*        STANDARD EXIT LINKAGE, AND EXIT FROM THIS MODULE             *
*                                                                     *
***********************************************************************
FINISHED EQU   *
         L     14,12(REG13)             RESTORE CALLER'S
         LM    0,12,20(REG13)           REGISTERS
         BR    REG14                    RETURN TO CALLER
         EJECT
***********************************************************************
*        CONSTANTS                                                    *
***********************************************************************
         DS    0F
ZERO     DC    F'0'                     ZERO
FOUR     DC    F'4'                     FOUR
         EJECT
         IXCYERE                        ERE
         EJECT
         END   ELEMEXIT
```

Chapter 42. IXC_WORK_RESTART — Workload Restart Exit

Topics for This Exit Appear as Follows:

Through the IXC_WORK_RESTART exit, your installation can prepare a system to receive additional workload from a failing system in the sysplex. MVS invokes IXC_WORK_RESTART one time on each system that is selected to restart work from a failing system. MVS selects the system most capable of handling the additional work. Because of the system's resources or unusual workload, your installation might want to improve this system's capability. Your installation can do so by coding the workload restart exit to perform tasks such as cancelling lower priority work.

This exit cannot cancel or change the restart of an element. To prevent the restart of an element, or to change how an element will be restarted, see Chapter 41, "IXC_ELEM_RESTART — Element Restart Exit," on page 289.

Installing the Exit Routine

IXC_WORK_RESTART must be a load module in LPA or in an APF-authorized library in the LNKLST concatenation, on all the systems in the sysplex that are, or may be, connected to the automatic restart management (ARM) couple data set. This exit must be linkedited as reentrant.

For general instructions on installing a dynamic exit routine, see "Link editing a Dynamic Exit Routine into a Library" on page 6.

Exit Routine Environment

The workload restart exit receives control on the system where the work is to be restarted, in the following environment:

- Task mode
- Supervisor state with PSW key 0
- 31-bit addressing mode
- Primary ASC mode
- Enabled for interrupts
- No locks held

Exit Recovery: Your installation must provide recovery for this exit, if it is critical for the exit to successfully complete its processing. If you do not provide recovery and the exit ends abnormally, MVS restarts proceed without modification.

Exit Routine Processing

MVS invokes the IXC_WORK_RESTART exit once on each system where elements from a failed system are to be restarted, just prior to any restarts. MVS passes to the exit a parameter list that contains information about the failing system and the elements to be restarted. Design this exit to use the parameter list information, and information about this system and its current workload and resources, to cancel lower priority work or take any other actions necessary to lessen the impact of or prepare for additional work on this system.

You cannot use IXC_WORK_RESTART to cancel or redirect the elements to another system. MVS expects no information on return from this exit, and so will not alter the restarts.

Programming Considerations

- Code IXC_WORK_RESTART to be reentrant.
- Because MVS expects no information on return from the exit, you might want to code the exit to issue a message to indicate successful or unsuccessful processing. You might also code it to issue messages to report whatever actions the exit has taken to prepare the system for the restarts.

Entry Specifications

MVS passes to IXC_WORK_RESTART the address of the parameter list mapped by IXCYWRE.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Does not contain any information for use by the exit
1	Address of the IXCYWRE parameter list
2-12	Does not contain any information for use by the exit
13	Register save area
14	Return address
15	Entry point address of IXC_WORK_RESTART

Parameter List Contents: Register 1 contains the address of the workload restart exit parameter list, which is mapped by the IXCYWRE mapping macro. The parameter list contains:

- The name of the failing system.
- The number of elements that are to be restarted on this system.
- The names of the elements that are to be restarted on this system.

IXCYWRE is described in *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Return Specifications

No information is expected on return to MVS.

Registers at Exit: MVS does not use any register contents returned from this exit, so you are not required to place any specific contents in the registers.

Coded Example of the Exit Routine

The following is an example of an IXC_WORK_RESTART exit.

```
TITLE 'WORKLOAD - SAMPLE AUTOMATIC RESTART MANAGER WORKLOAD RESTART EXIT'
***START OF SPECIFICATIONS*********************************************
*                                                                     *
* MODULE NAME         =  WORKLOAD                                     *
*                                                                     *
* DESCRIPTIVE NAME    =  SAMPLE AUTOMATIC RESTART MANAGER WORKLOAD    *
*                        RESTART EXIT.                                *
*                                                                     *
* FUNCTION            =  BASED ON THE NUMBER OF ELEMENTS TO BE        *
*                        RESTARTED, THIS EXIT WILL CANCEL THE SAME    *
*                        NUMBER OF JOBS ON THE SYSTEM.                *
*                                                                     *
*   OPERATION         =  DETERMINES THE NUMBER OF ELEMENTS TO BE      *
*                        RESTARTED FROM THE INPUT PARAMETER LIST.  IF *
*                        THE NUMBER IS LARGER THAN 10, 10 JOBS WILL BE*
*                        CANCELLED.  OTHERWISE, THE NUMBER OF JOBS TO *
*                        BE CANCELLED WILL BE THE SAME AS THE NUMBER  *
*                        TO BE RESTARTED.                             *
*                                                                     *
*                        THE JOB NAMES WILL BE OBTAINED FROM AN       *
*                        INTERNAL LIST AND EACH WILL BE CANCELLED VIA *
*                        THE MGCRE MACRO.  A WTO WILL BE ISSUED       *
*                        INDICATING THAT EACH JOB IS BEING CANCELLED. *
*                                                                     *
* ENTRY POINT         =  WORKLOAD                                     *
*                                                                     *
*   PURPOSE           =  TO FREE RESOURCES BY CLEANING UP LOW PRIORITY*
*                        WORK ON THE SYSTEM PRIOR TO RESTART.         *
*                                                                     *
*   LINKAGE           =  BALR                                         *
*                                                                     *
*   INPUT DATA        =  REG1  ADDRESS OF THE IXCYWRE PARAMETER LIST  *
*                        REG13 ADDRESS OF STANDARD SAVE AREA          *
*                        REG14 RETURN ADDRESS                         *
*                        REG15 ENTRY POINT ADDRESS                    *
*                                                                     *
*   REGISTERS SAVED   =  REG0 - REG15                                 *
*                                                                     *
*   REGISTER USAGE    =  REG0  - PARAMETER REGISTER                   *
*                        REG1  - PARAMETER REGISTER                   *
*                        REG2  - NOT USED                             *
*                        REG3  - WORK REGISTER                        *
*                        REG4  - WORK REGISTER                        *
*                        REG5  - POINTER TO WRE                       *
*                        REG6  - WORK REGISTER                        *
*                        REG7  - WORK REGISTER                        *
*                        REG8  - WORK REGISTER                        *
*                        REG9  - RETURN REGISTER FOR SUBROUTINES      *
*                        REG10 - NOT USED                             *
*                        REG11 - MODULE DATA REGISTER                 *
*                        REG12 - MODULE BASE REGISTER                 *
*                        REG13 - POINTER TO A STANDARD SAVE AREA      *
*                        REG14 - RETURN POINT                         *
*                        REG15 - PARAMETER REGISTER                   *
```

```
*                                                                     *
*   REGISTERS RESTORED = REG0 - REG15                                 *
*                                                                     *
*   CONTROL BLOCKS    =                                               *
*     NAME     MAPPING MACRO   REASON USED                  USAGE     *
*     ----     -------------   -----------                 -------    *
*     WRE       IXCYWRE         EXIT PARAMETER LIST          R       *
*                                                                     *
*   KEY = R-READ, W-WRITE, C-CREATE, D-DELETE                         *
*                                                                     *
*   TABLES            =  JOB NAME TABLE                               *
*                                                                     *
*   MACROS            =  GETMAIN, FREEMAIN, MGCRE, WTO                *
*                                                                     *
*   MESSAGES          =  USER001I                                     *
*                                                                     *
*   MODULE TYPE       = CSECT                                         *
*                                                                     *
*   ATTRIBUTES        =  REENTRANT, REUSABLE, AMODE 31, RMODE ANY     *
*                                                                     *
***********************************************************************
         EJECT
WORKLOAD CSECT
WORKLOAD AMODE 31                     31-BIT ADDRESSING MODE
WORKLOAD RMODE ANY                    31-BIT RESIDENCE
         SPACE 1
***********************************************************************
*                                                                     *
*         REGISTER ASSIGNMENTS                                        *
*                                                                     *
***********************************************************************
REG0     EQU   0                      REGISTER 0
REG1     EQU   1                      REGISTER 1
REG2     EQU   2                      REGISTER 2
REG3     EQU   3                      REGISTER 3
REG4     EQU   4                      REGISTER 4
WREPTR   EQU   5                      REGISTER 5 - POINTS TO WRE
REG6     EQU   6                      REGISTER 6
REG7     EQU   7                      REGISTER 7
REG8     EQU   8                      REGISTER 8
REG9     EQU   9                      REGISTER 9
REG10    EQU   10                     REGISTER 10 - CURRENTLY UNUSED
DATAREG  EQU   11                     REGISTER 11 - DYNAMIC DATA AREA
REG12    EQU   12                     REGISTER 12
BASEREG  EQU   12                     REGISTER 12 - MODULE BASE
REG13    EQU   13                     REGISTER 13
REG14    EQU   14                     REGISTER 14
REG15    EQU   15                     REGISTER 15
         SPACE 1
***********************************************************************
*                                                                     *
*        EQUATES                                                      *
*                                                                     *
***********************************************************************
SPINPRVT EQU   230                    SUBPOOL VALUE FOR GETMAIN
         EJECT
***********************************************************************
*                                                                     *
*        STANDARD ENTRY LINKAGE                                       *
*                                                                     *
***********************************************************************
         STM   REG14,REG12,12(REG13)    SAVE CALLER'S REGISTERS
         BALR  BASEREG,REG0             ESTABLISH MODULE BASE
         USING *,BASEREG                ESTABLISH ADDRESSABILITY
         LR    WREPTR,REG1              ESTABLISH ADDRESSABILITY
         USING WRE,WREPTR               TO THE WRE
***********************************************************************
```

```
*                                                                     *
*        OBTAIN DYNAMIC STORAGE                                       *
*                                                                     *
***********************************************************************
         SPACE 1
         LA    REG0,DATAEND                 LENGTH OF DATA AREAS
         GETMAIN  RU,LV=(REG0),SP=SPINPRVT  OBTAIN DYNAMIC STORAGE
         LR    DATAREG,REG1             ADDRESS RETURNED IN REG1
         USING DATAAREA,DATAREG         ADDRESSABILITY TO DYNAMIC      X
                                        STORAGE
***********************************************************************
*                                                                     *
*        DETERMINE THE NUMBER OF JOBS TO CANCEL BASED ON THE INPUT    *
*        NUMBER OF ELEMENTS TO BE RESTARTED.  IF THE NUMBER IS        *
*        LARGER THAN 10, 10 WILL BE USED.                             *
*                                                                     *
***********************************************************************
         SPACE 1
         LA    REG3,10                  MAX NUMBER OF JOBS TO CANCEL
         L     REG4,WRENUMBEROFELEMENTS NUMBER OF ELEMENTS TO RESTART
         CR    REG4,REG3                IS NUMBER GREATER THAN MAX?
         BL    CANCEL                   NO, GO TO CANCEL
         LR    REG4,REG3                USE THE MAX
***********************************************************************
*                                                                     *
*        CANCEL THE JOBS SPECIFIED IN THE INTERNAL TABLE VIA THE      *
*        MGCRE MACRO.  ISSUE A WTO INDICATING EACH JOB THAT IS        *
*        CANCELLED.                                                   *
*                                                                     *
***********************************************************************
CANCEL   EQU   *
         ST    REG4,JOBNUM              SAVE THE NUMBER OF JOBS
         LA    REG3,0                   SET COUNTER TO ZERO
LOOP     EQU   *
         LR    REG7,REG3                MANIPULATE COUNTER VALUE TO
         M     REG6,ENTRYLNG            OBTAIN JOBS INDEX VALUE
         LA    REG6,JOBS                GET ADDRESS OF JOBS TABLE
         AR    REG7,REG6                ADD THE DISPLACEMENT
         MVC   JOBNAME(8),0(REG7)       STORE JOB NAME TO BE CANCELLED
         BAL   REG9,ISSMGCRE            ISSUE CANCEL COMMAND
         BAL   REG9,ISSUWTO             ISSUE MESSAGE USER001I
         A     REG3,ONE                 INCREMENT COUNTER BY ONE
         C     REG3,JOBNUM              IF COUNTER IS LESS THAN NUMBER
         BL    LOOP                     OF ELEMENTS, LOOP
         EJECT ,
***********************************************************************
*        RETURN THE DYNAMIC STORAGE OBTAINED                          *
***********************************************************************
         LA    REG0,DATAEND                 LENGTH OF DATA AREAS
         FREEMAIN  RU,LV=(REG0),A=(DATAREG),SP=SPINPRVT               X
                                       FREE THE DYNAMIC STORAGE AREA
***********************************************************************
*                                                                     *
*        STANDARD EXIT LINKAGE, AND EXIT FROM THIS MODULE             *
*                                                                     *
***********************************************************************
FINISHED EQU   *
         LM    REG14,REG12,12(REG13)    RESTORE CALLER'S              X
                                        REGISTERS
         BR    REG14                    RETURN TO CALLER
         EJECT
***********************************************************************
*                                                                     *
*   PROCEDURE - ISSMGCRE                                              *
*                                                                     *
*   FUNCTION  - ISSUES A CANCEL COMMAND FOR A JOB                     *
```

```
*                                                                     *
*   INPUT     - REGISTER 11 POINTS TO THE DYNAMIC AREA WHICH CONTAINS *
*               STORAGE FOR THE MGCRE PARAMETER LIST                  *
*                                                                     *
*   OUTPUT    - A CANCEL COMMAND IS ISSUED VIA MGCRE                  *
*                                                                     *
*   NOTES     - LIST AND EXECUTE FORMS OF MGCRE ARE REQUIRED.         *
*                                                                     *
***********************************************************************
         SPACE 1
ISSMGCRE EQU   *
         LA    REG7,DYNCMDR1            ADDRESS COMMAND REPLY AREA
         MVI   0(REG7),BLANK            BLANK FIRST MESSAGE CHARACTER
         MVC   1(CMDRLENG-1,REG7),0(REG7) BLANK ENTIRE MESSAGE FIELD
         MVC   0(L'CMDLNGTH,REG7),CMDLNGTH  INITIALIZE COMMAND LENGTH
         MVC   L'CMDLNGTH(L'TXINSRT1,REG7),TXINSRT1
*                                     MOVE CANCEL VERB TO AREA
***********************************************************************
*        JOBNAME CONTAINS THE JOB NAME VALUE. ENTRYLNG                *
*        CONTAINS THE LENGTH OF THE JOB NAME. USE THESE FIELDS TO     *
*        MOVE THE JOB NAME INTO THE TEXT AREA                         *
***********************************************************************
         LH    REG8,ENTRYLNG            JOB NAME LENGTH
         BCTR  REG8,0                   DECREMENT BY 1 FOR EXECUTE
         EX    REG8,MOVEJOBN            MOVE JOB NAME VALUE INTO TEXT
         LA    REG4,DYNMGCRE            ADDRESS MGCRE PARAMETER LIST
         MVC   0(REPLEN,REG4),REPAREA   COPY MGCRE LIST TO DYNAMIC
         XR    REG6,REG6                CONSOLE ID VALUE OF ZERO
         MGCRE TEXT=(REG7),CONSID=(REG6),MF=(E,(REG4))
         BR    REG9                     RETURN TO CALLER
***********************************************************************
*        OBJECT OF AN EXECUTE                                         *
***********************************************************************
MOVEJOBN MVC   TXINSRT2-CMDCANCL(0,REG7),JOBNAME   MOVE JOB NAME
*                                     INTO CANCEL COMMAND
         EJECT
***********************************************************************
*                                                                     *
*   PROCEDURE - ISSUWTO                                               *
*                                                                     *
*   FUNCTION  - ISSUES A MESSAGE INFORMING OPERATOR THAT A JOB IS     *
*               BEING CANCELLED                                       *
*                                                                     *
*   INPUT     - NONE                                                  *
*                                                                     *
*   OUTPUT    - MESSAGE STATING JOB CANCELLED                         *
*                                                                     *
***********************************************************************
         SPACE 1
ISSUWTO  EQU   *
***********************************************************************
*        INITIALIZE MESSAGE TEXT FOR TEXT= PARAMETER ON WTO           *
***********************************************************************
         LA    REG8,DYNAMTXT            DYNAMIC MESSAGE AREA
         MVC   0(USERLENG,REG8),USERMSG1 INITIALIZE MESSAGE TEXT
         MVC   USERM1VB-USERMSG1(L'USERM1VB,REG8),JOBNAME   JOB NAME
         MVC   DYNUSERS,USERSTAT        MOVE WTO STATIC AREA
         SR    REG0,REG0                INITIALIZE REGISTER 0 TO ZERO
         WTO   TEXT=(REG8),ROUTCDE=(11),DESC=(6),MF=(E,DYNUSERS)
*                                       ISSUE THE MESSAGE
         BR    REG9                     RETURN TO CALLER
         EJECT
***********************************************************************
*                                                                     *
*        JOB NAME TABLE                                               *
*                                                                     *
***********************************************************************
```

```
JOBS     DC    CL8'JOB1    '
         DC    CL8'JOB2    '
         DC    CL8'JOB3    '
         DC    CL8'JOB4    '
         DC    CL8'JOB5    '
         DC    CL8'JOB6    '
         DC    CL8'JOB7    '
         DC    CL8'JOB8    '
         DC    CL8'JOB9    '
         DC    CL8'JOB10   '
         SPACE 1
*********************************************************************
*        LENGTH AND TEXT OF CANCEL COMMAND                          *
*********************************************************************
CMDCANCL EQU   *
CMDLNGTH DC    XL2'0F'                 LENGTH OF CANCEL COMMAND
TXINSRT1 DC    CL7'CANCEL '            FIRST STATIC INSERT
TXI1LENG EQU   *-CMDCANCL              OFFSET TO TXINSRT1
TXINSRT2 DC    CL8'XXXXXXXX'           PLACE JOB NAME HERE
CMDRLENG EQU   *-CMDCANCL              COMMAND LENGTH
         EJECT ,
*********************************************************************
*                                                                   *
*        LIST FORM OF MGCRE MACRO (STATIC)                          *
*                                                                   *
*********************************************************************
USERREP  DS    0H
REPAREA  MGCRE MF=L                    LIST FORM OF MACRO
REPLEN   EQU   *-USERREP               LENGTH OF MGCRE PARAMETER LIST
*********************************************************************
*                                                                   *
*        LIST FORM OF WTO MACRO (STATIC)                            *
*                                                                   *
*********************************************************************
USERSTAT DS    0H
         WTO TEXT=USERMSG1,ROUTCDE=(11),DESC=(6),MF=L
CNCLMSGL EQU   *-USERSTAT               LENGTH OF PARAMETER LIST
         SPACE 1
*********************************************************************
*        CONSTANTS                                                  *
*********************************************************************
         DS    0F
ENTRYLNG DC    F'8'                    LENGTH OF JOB NAME TABLE ENTRY
ONE      DC    F'1'                    ONE
BLANK    EQU   X'40'                   BLANK CHARACTER
         SPACE 1
*********************************************************************
*        LAYOUT OF MESSAGE TEXT FOR USER001I                        *
*********************************************************************
USERMSG1 DS    0H
USERM1LN DC    XL2'1C'                 LENGTH OF MESSAGE (IN HEX)
USERM1S1 DC    C'USER001I CANCELLING '  FIRST STATIC FIELD
USERM1VB DC    CL8' '                  JOB NAME
USERLENG EQU   *-USERMSG1              LENGTH OF AREA FOR GETMAIN
         EJECT
*********************************************************************
*                                                                   *
*        STORAGE DEFINITIONS                                        *
*                                                                   *
*********************************************************************
DATAAREA DSECT
         DS    0F
JOBNUM   DS    F
JOBNAME  DS    CL8
DYNMGCRE DS    CL(REPLEN)              DYNAMIC MGCRE AREA
DYNAMTXT DS    CL(USERLENG)            DYNAMIC MESSAGE TEXT AREA
DYNUSERS DS    CL(CNCLMSGL)            WTO MACRO AREA
```

```
DYNUMSG1 DS    CL(USERLENG)            DYNAMIC USER001I MESSAGE
DYNCMDR1 DS    CL(CMDRLENG)            DYNAMIC COMMAND REPLY AREA
         DS    0H
         ORG
DATAEND  EQU   *-DATAAREA
         EJECT
         IXCYWRE                       WRE
         EJECT
         END   WORKLOAD
```

Chapter 43. IXGSEXIT — Log Stream Subsystem Exit

Topics for This Exit Appear as Follows:

If the log stream owner has special processing considerations, the log stream subsystem exit must be modified.

Note: The log stream subsystem exit can support applications that use QSAM and BSAM (GET and READ) requests in a sequential fashion. The exit cannot support applications that attempt to use other access method services when processing the records maintained in a log stream. If other access method services are required by the application or if the application does not intend to obtain records in a sequential fashion, you will need to update the application to make use of the system logger services. QSAM and BSAM access methods support approximately 32K record sizes. A log stream log block can have record lengths of up to 64K-4. If log blocks (no blocking) or individual records are written with a size greater than what the access methods allow, then applications using the LOGR subsystem will not be able to obtain entire records.

The log stream subsystem exit receives control at six different points in the processing of an application's JCL DD statement that has the SUBSYS=(LOGR,exit_routine_name,...) specification. Each point causes the exit routine to be invoked when the DD with the SUBSYS specification is encountered, with the exception of the GET request. The exit receives control during GET processing once per GET request from the requesting application.

You can use the log stream subsystem exit to:

- At the converter exit point:
  - Validate the JCL parameters on the DD's SUBSYS keyword.
  - Cause system-type messages to be issued to the job's log.
  - Cause the job to end processing because of a JCL error.
- At the allocation exit point:
  - Validate the JCL parameters on the DD's SUBSYS keyword.
  - Cause system-type messages to be issued to the job's log.
  - Cause the job to end processing because of a JCL error.

  - Establish data areas or obtain resources that will be persistent throughout the other exit points.
- At the open exit point:
  - Connect to the system logger log stream.
  - Start the log stream browse session.
- At the get exit point:
  - Return a record to the requesting application.
- At the close exit point:
  - End the log stream browse session.
  - Disconnect from the system logger log stream.
- At the unallocation exit point:
  - Cause unneeded log blocks to be deleted from the log stream.
  - Disconnect from the system logger log stream.
  - Return data areas or resources that were obtained in prior exit event calls.

Installing the Exit

The log stream subsystem exit must be linkedited in its own load module into SYS1.LINKLIB or any APF-authorized library in the LNKLST concatenation. To activate the exit routine, refresh LLA through the MODIFY LLA,REFRESH command.

The name of the exit is determined by the log stream owner and must match the value in the second positional parameter of the SUBSYS=(LOGR,exit_routine_name,...) DD JCL specification for applications that intend to access records from the owner's log stream. The name of the exit must also match its load module and entry point names. It is the responsibility of the log stream owner to provide the exit routine name to applications that will use the exit. If an exit routine name is not specified on the SUBSYS=(LOGR,exit_routine_name,...) statement, IXGSEXIT will be used as a default exit.

Note: To use the log stream subsystem exit, the LOGR subsystem must be activated. See *z/OS MVS Setting Up a Sysplex* for more information.

Exit Routine Environment

The log stream subsystem exit receives control in the following environment:

- Task mode
- Enabled for interrupts and unlocked
- Primary ASC mode

As Table 5 on page 305 shows, other environmental factors for the log stream subsystem exit routine are based on the event that causes the routine to be invoked.

Table 5. Environmental factors for the log stream subsystem exit routine

Event	Authorization	AMODE	Cross Memory Mode	Restrictions
Converter	Supervisor state with PSW key 1	31	PASN = HASN, any SASN	The control blocks passed to the exit routine do not reside in the application's address space. The application's address space might even be on a different processor than where converter processing is occurring.
Allocation	Supervisor state with PSW key 1	31	PASN = HASN, any SASN	The control blocks passed to the exit routine reside in the application's address space.
Unallocation	Supervisor state with PSW key 1	31	PASN = HASN, any SASN	The control blocks passed to the exit routine reside in the application's address space.
Open	Supervisor state with PSW key 1	31	PASN = HASN, any SASN	The control blocks passed to the exit routine reside in the application's address space.
Close	Supervisor state with PSW key 1	31	PASN = HASN, any SASN	The control blocks passed to the exit routine reside in the application's address space.
Get	Problem state with PSW key of the application (typically key 8)	31	PASN = HASN, any SASN	The control blocks passed to the exit routine reside in the application's address space.

Linkage Conventions: The log stream subsystem exit is invoked via BALR R14,R15 for each event or purpose.

Exit Recovery: The log stream subsystem exit routine should provide its own recovery. If it does not provide a recovery routine, or if an exit routine error percolates beyond the exit's recovery, the system's ESTAEX recovery routine will get control. The ESTAEX will record information in the SDWA and request an SDUMP.

If the exit abends, the system will not allow the current point to continue. Based on the specific point, an abend in the exit may cause the job or application to end. If the exit abends, however, and the log stream owner provides recovery that returns to the normal return point of the invoking routine, the LOGR subsystem will not cause the job or application to end.

Exit Routine Processing

The processing in the exit depends on which point caused the exit to be invoked.

The exit is invoked with register 1 pointing to a word that contains the address of the common parameter list for the exit (IXGSXCMP). Code your exit routine to check IXGSXCMP_EVENT for the type of call for which it was invoked. The IXGSXCMP parameter list contains common information to all the exit calls. In addition, field IXGSXCMP_SPECIFIC_PTR points to the specific point's parameter list extension.

Converter Call

The log stream subsystem exit receives control during JCL converter processing each time a DD statement containing a SUBSYS=(LOGR,exit_routine_name,...) is encountered. The purpose of this call is to validate the JCL parameters. It is not meant to build any control blocks based on the JCL parameters; because this processing runs in a different address space, possibly on a different system from the actual job processing, any control blocks built during this call would not be accessible during later calls.

Prior to the log stream subsystem exit getting control, the LOGR subsystem validates and checks the syntax of the SUBSYS parameters with the exception of *SUBSYS-options2*. If there are any errors, an error return code is returned to the converter. Otherwise, the log stream subsystem exit is invoked.

When the log stream subsystem exit routine receives control, it can also validate the input JCL, especially the *SUBSYS-options2* parameters. The *SUBSYS-options2* parameters are unique to each exit, and so the exit must check for syntax errors.

If there are no errors, the job continues processing. If a parse error occurs, the job ends. If the system provided an error message describing the parse error, the message will be returned so that it can be placed in the job log. If an abend occurs, the job ends.

Allocation Call

The log stream subsystem exit receives control during allocation processing each time a DD statement containing a SUBSYS=(LOGR,exit_routine_name,...) is encountered. The exit can perform similar processing as for the converter call (dynamic allocation does not go through converter processing). You can set field IXGSXCMP_EXIT_TOKEN to be used as input for the other exit calls.

Prior to the log stream subsystem exit getting control, the LOGR subsystem validates and checks the syntax of the SUBSYS parameters with the exception of the *SUBSYS-options2*. If there are any errors it will return an error return code to allocation. Otherwise it will invoke the log stream subsystem exit routine.

When the log stream subsystem exit routine receives control, it can also validate the input JCL, especially the *SUBSYS-options2* parameters. The *SUBSYS-options2* parameters are unique to each exit, and so they are the exit's responsibility for syntax checking.

If there are no errors, the job continues processing. If a parse error occurs, the job ends. If the system provided an error message describing the parse error, the message will be returned so that it can be placed in the job log. If an abend occurs, the job ends.

Open Call

The log stream subsystem exit receives control during OPEN processing each time a DD statement containing a SUBSYS=(LOGR,exit_routine_name,...) is encountered.

When the log stream subsystem exit routine receives control, it might ensure that the log stream can be accessed by the current job or application. When a log stream connect is issued using the IXGCONN macro, system logger processing will

perform the SAF authorization checking. The *xstreamname* to be used on the connect can be obtained from field IXGSXCMP_LOGNAME in data area IXGSXCMP.

After the log stream is connected, the log stream subsystem exit can start a browse session. See *z/OS MVS Programming: Assembler Services Guide* for more information about a browse session. The SUBSYS parameters have been processed by the LOGR subsystem and the values have been corrected in fields in the IXGSXCMP parameter list. If some of these keywords are not specified, defaults were used. Logger can add new parameters on the SUBSYS-options1 set. New parameters that are added to this set are done in a compatible manner. It is up to the exit routine owner to provide additional support to take advantage of the new options. For example, the VIEW= ACTIVE | ALL | INACTIVE options were newly added as maintenance on HBB6608 and higher releases and on JBB7713.

The exit needs to save the log stream connect token and browse token for use on GET exit calls to browse the log stream data. The exit needs to save the browse token for use on GET exit calls.

The name of the routine to process the GET requests can be provided in the IXGSXOCP parameter list. This allows a separate exit routine to be invoked and operate in problem program state and key 8. It might make the exit routine's structure and processing easier to provide a separate routine to handle the GET requests.

When the exit returns to the LOGR subsystem, control returns to OPEN processing. If there are no errors, the job continues. If the request was unsuccessful, DFSMS issues an abend for the open failure.

Get Call

The log stream subsystem exit receives control during GET processing each time a GET request from the requesting application containing a SUBSYS=(LOGR,exit_routine_name,...) is encountered. The intent of this exit call is to allow a log stream owner the ability to return records from a log stream as if they were being obtained from a conventional data set. This is for QSAM or BSAM requests only.

On GET requests, the LOGR subsystem sets up the input parameter list and passes control to the log stream subsystem exit. The log stream subsystem exit receives control in the application's state and key (typically problem state, key 8).

The exit routine obtains log stream blocks, deblocks and formats the logical records, if appropriate, and returns them to the application.

After the exit routine returns, the LOGR subsystem returns control to GET processing with a normal or error return code.

Close Call

The log stream subsystem exit receives control during CLOSE processing each time a DD statement containing a SUBSYS=(LOGR,exit_routine_name,...) is encountered.

When the log stream subsystem exit routine receives control, it might end the log stream browse session established during the Open exit call.

When the exit returns to the LOGR subsystem, control returns to CLOSE processing. If there are no errors, the job continues. If the request was unsuccessful, DFSMS issues an abend for the close failure.

Unallocation Call

The log stream subsystem exit receives control during Unallocation processing each time a DD statement containing a SUBSYS=(LOGR,exit_routine_name,...) is encountered.

The exit might need to invoke other system logger functions prior to disconnecting from the log stream. For example, the exit can invoke the IXGDELET service to delete log stream blocks from the oldest to the block just before a specified blockid. Optionally, all the log stream blocks can be deleted on the request.

The exit should ensure that all resources that were obtained during the other exit calls are returned to the system. For example,

```
IXGCONN REQUEST=DISCONNECT,STREAMNAME=xstreamname,
        STREAMTOKEN=xstreamtoken,...
```

The *xstreamname* to be used on the disconnect can be obtained from field IXGSXCMP_LOGNAME in data area IXGSXCMP. The *xstreamtoken* is the value returned on the previous log stream connect request for this DD statement.

If the exit used field IXGSXCMP_EXIT_TOKEN as an anchor for keeping persistent data across the exit calls, then the storage should be returned to the system.

If there are no errors, the job continues.

Programming Considerations

If the log stream owner does not have any special processing requirements for handling the SUBSYS parameters and interfacing with the system logger, the system logger default exit routine, IXGSEXIT, may be suitable to satisfy the processing requirements.

You might not want to use IXGSEXIT. You might want to code your own exit routine, for any number of reasons, including:

- Data read from the log stream need to be de-blocked. For example, data might now be written to a log stream in log blocks that contains more than one record of data per block. IXGSEXIT returns only one entire block of data to the caller for each read request.
- Additional processing features are required. IXGSEXIT supports only the FROM, TO, DURATION, and VIEW keywords of the LOGR subsystem. See *z/OS MVS Programming: Assembler Services Guide* for more information about the LOGR subsystem keywords.
- IXGSEXIT tries again when a system logger service returns a log stream or a resource-temporarily-unavailable condition.

Consider the following scenario, in which you are given three options. You have an application that writes data to one data set per system in a sysplex. Some users of your application have written tools to read the data using QSAM/BSAM.

After installing MVS 5.2, you change your application to write to a single log stream instead of to system data sets.

You now have three choices:

- Inform your users that they have to write new code and use system logger services to read the data.
- Inform your users that they have to change their JCL and use the LOGR subsystem with default exit IXGSEXIT.
- Write a log stream subsystem exit with more features than IXGSEXIT, and inform your users to change their JCL to use the LOGR subsystem with your exit.

If you decide to write your own log stream subsystem exit, code the exit routine to be reentrant. A new copy of the exit is loaded into storage from SYS1.LINKLIB for each application that specifies the appropriate SUBSYS parameters to obtain records from a log stream.

- Converter Call

  The conversion of a job's JCL will take place in a different address space and potentially on a different system image from where the application will run. The exit routine should not attempt to keep persistent data from this invocation of the exit to the other invocations of the exit.

- Allocation Call

  The allocation invocation of the exit provides a mechanism for the exit to obtain persistent data that can be used in the other invocations of the exit.

  Field IXGSSCMP_EXIT_TOKEN can be set by the log stream subsystem exit routine during the allocation call and it will then be input to the next exit call for this DD statement.

  It is the responsibility of the exit routine to ensure that the resources it obtains are released. If the exit routine obtains any resources that are not explicitly job related, such as common storage, a resource manager routine may need to be established. If an abnormal memory end occurs, close and unallocation processing do not occur, so the exit routine will not receive control through this exit interface.

  Note: Log stream data sets can be concatenated. No special processing by the exit is necessary to handle this other than recognizing that the DDname (field IXGSXAP_DDNAME in the IXGSXAP data area) can be blank. The system automatically goes through open and close functions for each data set although the application does not issue them for each data set.

- Open Call

  Field IXGSSCMP_EXIT_TOKEN can be used to pass data from the Allocation call to the Open exit routine. It can also be set as output from the Open exit routine to pass data to the subsequent log stream subsystem exit invocations.

  It is the responsibility of the log stream subsystem exit to establish the correct connection and references to the log stream.

  It is possible that Logger will add new parameters on the SUBSYS-options1 set. New parameters that are added to this set are done in a compatible manner. It is up to the exit routine owner to provide additional support to take advantage of the new options. For example, the VIEW= ACTIVE | ALL | INACTIVE options were newly added as maintenance on HBB6608 and higher releases and on JBB7713.

  The exit needs to save the log stream connect token and browse token for use on GET exit calls to browse the log stream data.

  An end of file condition may be encountered via a specific return/reason code from the IXGBRWSE API or as a result of the exit routine determining that no

data meets the input criteria. If the exit routine returns an error condition on the Open call, then DFSMS will fail/ABEND the DD OPEN request. To avoid the OPEN failure condition, the exit routine could indicate the end of file condition in its persistent data area and use that indication on the first GET call. This allows the program that is attempting to read the log stream data to simply receive an end of file (or end of data) condition vs. an OPEN failure.

- Get Call

  Any special deblocking or formatting of records from a log stream block needs to be performed during this call.

  The GET exit routine should recognize any special conditions established during the Open call exit routine processing.

- Close Call

  It is possible for an application to not issue an OPEN for a DD that was allocated. In addition, a DD that was opened cannot be explicitly closed by the application. The exit routine must ensure that it recognizes these conditions and returns resources to the system accordingly.

- Unallocation Call

  When the exit routine is invoked for unallocation, all resources that were obtained by the exit should be returned.

  It is the responsibility of the exit routine to ensure that the resources it obtains are released. If the exit routine obtains any resources that are not explicitly job related, such as common storage, a resource manager routine may need to be established. If an abnormal memory end occurs, close and unallocation processing do not occur, so the exit routine will not receive control through this exit interface.

  It is possible for an application to not issue an OPEN for a DD that was allocated. In addition, a DD that was opened may not be explicitly closed by the application. The exit routine must ensure that it recognizes these conditions and returns resources to the system accordingly.

Entry Specifications

The LOGR subsystem passes a standard format parameter list and linkage on entry to the log stream subsystem exit routine.

Registers at Entry: The contents of the registers on entry to the exit are as follows.

Register	Contents
0	Not applicable
1	Pointer to a full word field containing the address of the log stream subsystem exit common parameter list (IXGSXCMP). The high-order bit in the full word field pointed to by register 1 is set on ('1'b) to indicate the end of the input parameter list.
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit

The contents of the registers on entry to this exit are: See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for the

following data areas, which are used as parameter lists: IXGSXCMP, IXGSXCNP, IXGSXAP, IXGSXOCP, IXGSXGP, IXGSXUP, IXGSXMSP, and IXGSXTXT.

Return Specifications

The exit can return information in specific output fields in the parameter list and in register 15 to indicate a return code.

Output Parameter Descriptions, Converter are as follows.

IXGSXCNP_ISSUE_MSG
Issue message indicator. The message contained in the area pointed to by IXGSXCNP_MSG_PTR is to be issued as part of the system messages in the job's log. The message area contains a halfword length field followed by the message text. The length of the message text does not include the 2-byte prefix. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for data area IXGSXMSP.

Output Parameter Descriptions, Allocation are as follows

IXGSXCMP_EXIT_TOKEN
Exit token. The value returned in this field will be provided as input to the next exit call for this DD.

IXGSXAP_ISSUE_MSG
Issue message indicator. The message contained in the area pointed to by IXGSXAP_MSG_PTR is to be issued as part of the system messages in the job's log (during batch allocation). The message area contains a halfword length field followed by the message text. The length of the message text does not include the 2-byte prefix. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for data area IXGSXMSP.

IXGSXAP_INFO_CODE
DD error information code. Passed back to allocation to identify the reason why the allocation for the DD statement failed.

Output Parameter Descriptions, Open are as follows.

IXGSXCMP_EXIT_TOKEN
Exit token. The value returned in this field will be provided as input to the next exit call for this DD statement.

IXGSXOCP_IOEXIT_NAME
Name of the routine to be invoked on GET requests. This routine will be loaded into storage and called for the GET exit processing. The field is initialized with the exit name specified as the second positional parameter on the SUBSYS keyword.

Output Parameter Descriptions, GET are as follows.

IXGSXGP_RETURN_CODE
Return code to be passed to the issuer of the GET request. The following values, in decimal, can be set:

IXGSXGP_OK
0 - record is returned in the user's area.

IXGSXGP_LOGICAL_ERROR
8 - logical error was encountered. The record was not returned in the user's area.

IXGSXGP_LOGICAL_ERROR
24 - the exit had an abend or a system error and could not process the request. Do not continue job processing. The record was not returned in the user's area.

IXGSXGP_ERROR_CODE
Error code used to identify the reason for a non-zero value in IXGSXGP_RETURN_CODE. The following values, in decimal, can be set:

IXGSXGP_NO_ERROR
0 - no error was encountered. The field is initialized with a zero.

IXGSXGP_END_OF_DATA
4 - end of data condition was detected. The record was not returned in the user's area.

IXGSXGP_PERM_ERROR
8 - a permanent error condition was detected.

Output Parameter Descriptions, Close are as follows.

IXGSXCMP_EXIT_TOKEN
Exit token. The value returned in this field will be provided as input to the next exit call for this DD statement.

Output Parameter Descriptions, Unallocation are as follows.

IXGSXCMP_EXIT_TOKEN
Exit token. No longer applicable because unallocation is the last in the series of calls.

Registers at Exit: Upon return from the exit processing, the register contents must be as follows.

Register
Contents

0-1 Not relevant

2-14 Restored to contents at entry

15 One of the following return codes:

Return Code
Explanation

Value of 0
Job processing is to continue. An error message may be provided to be put into the system error message portion of the job's log. Refer to specific exit calls for details for these messages.

Value of 4
Job processing is not to continue. An error message may be provided to be put into the system error message portion of the job's log. Refer to specific exit calls for details for these messages.

Value of 20
The exit had an abend or logical error and could not process the request. Job processing is not to continue. An error message may

be provided to be put into the system error message portion of the job's log. Refer to specific exit calls for details for these messages.

Value other than 0 or 4
Treated the same as for return code 20.

Chapter 44. MVS Commands Installation Exit

Topics for This Exit Appear as Follows:

The MVS commands installation exit allows you to modify command processing in a system or sysplex. Use one or more MVS commands exit routines to modify command text or modify the MCS authority of consoles that issue commands (for example, to allow a console to issue a command for which it is not authorized).

You can use MVS commands installation exit routines to:

- Change the text of commands
- In a sysplex, change the destination of commands by routing them to a different system for execution
- Modify a console's MCS authority for a particular command. That is, you can use the exit to:
  - Allow the command from a console that normally would not have the MCS authority to issue the command
  - Reject the command from a console that normally would have the MCS authority to issue the command
- Execute commands
- Suppress commands

Installing the Exit Routine

You can insert MVS commands exit routines into the control program by:

- Linkediting the routines into an APF-authorized library as part of the LNKLST concatenation. Use 31-bit addresses in the routines and assemble them with AMODE 31. RMODE ANY is recommended.

- Specifying the name of each exit routine on the USEREXIT parameter of the .CMD statement of the required MPFLSTxx member of SYS1.PARMLIB. If you specify more than one exit routine, the routines are called in the order in which they are specified on the .CMD statement. The name of each exit routine can be from 1 to 8 alphanumeric characters.
- Activating the MPFLSTxx member with SET MPF=xx

For more information on how to specify MVS commands exit routines in the MPFLSTxx member of SYS1.PARMLIB, see *z/OS MVS Initialization and Tuning Reference*.

Replacing an MVS Commands Exit Routine Without a ReIPL: There may be times when you need to replace a commands exit routine, either because you want to add functions to the routine or because the routine abended when it was processing a particular command.

If you want to replace a commands exit routine with a fresh copy, you must do the following:

- Linkedit the new copy of the routine into SYS1.LINKLIB
- Refresh LLA with the MODIFY LLA,REFRESH command
- Reactivate the exit routine using the SET MPF=xx command

Deactivating a Command Exit: There are times when you might want to deactivate a command exit routine, perhaps because its function is not required at particular times or because you want to modify the routine. You can deactivate a command exit routine in one of two ways:

- Specify, on the .CMD statement of the required MPFLSTxx member, the name of a command exit that does not exist in SYS1.LINKLIB, such as 'USEREXIT(NONE)'. Enter the SET MPF=xx command to refresh the MPFLSTxx member.

  This action effectively deactivates any command exits that were enabled during the prior MPFLSTxx activation. The system issues an informational message that can be ignored in this case.
- Enter the SET MPF=NO command to disable all active MPFLSTxx members. Remove the exit name from the .CMD statement of the appropriate MPFLSTxx member and enter the SET MPF=xx command to resume MPF processing.

  Attention: Entering the SET MPF=NO command deactivates all installation-specified MPF options. IBM-supplied defaults are used until the installation reactivates its MPFLSTxx members.

Exit Routine Environment

MVS commands exit routines receive control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In primary ASC mode.
- With no locks held; they must return control with no locks held.
- In AMODE 31. RMODE ANY is recommended.
- In the address space of the routine that issued the command.

Exit Recovery: The MVS commands exit routines must provide their own level of recovery because, with one exception, the system does not continue to pass control

to an exit routine after it abnormally terminates. The exception is when the exit routine is to be deleted and the installation has provided a clean-up routine that will get control for termination calls. When the system calls the exits for deletion, a commands exit routine can invoke the clean-up routine to release any work areas the exits may have created.

See "Installing the Exit Routine" on page 315 for information on how to reactivate the exit routine if it abnormally terminates.

See "Communication Between the Exits" on page 319 for more information on exit routine work areas and clean-up routines.

Exit Routine Processing

The MVS commands exit routines get control whenever a command is issued. Command processing invokes the exit prior to issuing the subsystem interface (SSI) call for command processing.

The MVS commands exit routine parameter list (the CMDX) contains a command buffer (CMDXCLIB). The buffer contains the command text, and the length of the command text, to be processed as it was entered on the console. The exit must place the modified command text and its new length back into this buffer before it returns control to the calling module. The modified command text can be up to 126 bytes long.

Operator commands may contain the following characters:

- A to Z
- 0 to 9
- ' # $ & ( ) * + , - . / ¢ < | ! ; ¬ % _ > ? : @ " =

The system translates characters that are not valid into null characters (X'00').

If you are modifying the MCS authority of the console for this command in the exit routine, you must place the modified command authority into the CMDXAUTH field in the CMDX. Also you must set the appropriate authority change bits in the CMDXAFLA field in the CMDX.

The CMDX is mapped by the IEZVX101 macro (data area CMDX). See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for a description of the IEZVX101 mapping.

Changing Command Text with Exit Routines: If a command installation exit changes the text of a command, the system does the following:

- Logs the "new" text of the command (the result of the change by the exit routine)
- Issues message IEE295I to display both the original text and the new text.

If the command installation exit specified system symbols in the new command text, the system *does not* substitute text for those system symbols. The system symbols appear in the new command text in their original format. To add or change a system symbol in command text and have it processed, the exit can create a copy of the new system symbol text, and then call the ASASYMBM service explicitly to substitute text for the system symbol.

For more information about ASASYMBM,see *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. See the section on sharing system commands in *z/OS MVS System Commands* for more information about using system symbols in commands.

Considerations for System Symbols: When a command contains system symbols, MVS provides the command text to command installation exits *after* it substitutes text for the system symbols. For example, if the following command is entered to display a console group on system SYS1:

```
DISPLAY CNGRP,G=(CN1GRP&SYSCLONE.)
```

The command installation exit receives the following text (assuming that the default for &SYSCLONE., the last two characters of the system name, is taken):

```
DISPLAY CNGRP,G=(CN1GRPS1)
```

If a command installation exit requires the original command text (the one that existed *before* symbolic substitution), the exit can do the following:

1. Access the CMDXSYMS field in the CMDX to validate that the command was changed by symbolic substitution
2. If the command was changed by symbolic substitution, access the CMDXOLIP field in the CMDX to obtain the address of structure CMDXOLIB, which contains the original command text (before symbolic substitution occurred).

For a description of the IEZVX101 mapping macro, which maps the CMDX, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

MVS Commands Exit Routines in a Sysplex Environment: In a sysplex, a command can be routed from one system to another. If a .CMD statement specifies MVS commands exit routines in an active MPFLSTxx member for a system, the exit routines on that system are invoked. (For a description of the .CMD statement, see *z/OS MVS Initialization and Tuning Reference*.) Which system executes command exit routines depends on the following:

- If the ROUTE command is used, the MVS commands exit routines, if specified, are invoked on both the system that issued the ROUTE command and the system that received the routed command. However, the exit routines on the target system do not see 'ROUTE'.
- If a command is issued with an L= parameter, and the console addressed is on a different system, only the command exit routines on the system that issued the command are invoked. The exit routines on the system in which the console specified by L= is attached, are not invoked.
- If the MCS command prefix facility (CPF) is used to route a prefix command from one system to another system, only the command exit routines on the receiving system are invoked.
- If commands are directed to a specific system (via the CMDSYS option specified in the CONSOLxx member of SYS1.PARMLIB), only the command exit routines on the receiving system are invoked.

Programming Considerations

When you code an MVS commands exit routine, observe the following conventions:

- The MVS commands exit routines get control **before** the subsystem interface (SSI) passes control to subsystems enabled for function code 10 (command processing SSI call). For information on SSI function code 10, see *z/OS MVS Using the Subsystem Interface*.
- If you specify REMOVE=YES on the CPF macro, the system removes the command's prefix before invoking the MVS commands exit routines. If STRIP is specified for the prefix, the prefix will be stripped before the exit gets control.
- The MVS commands exit routines must be reentrant and serially reusable. Do not use macros with expansions that store information into an inline parameter list.
- Do not code an exit routine that receives control for a command that the routine issues; this causes an endless loop. The exit routine must be coded so that when it receives control for that command, it does not issue the command again.
- If you specify command text, or a text length value, that exceeds the maximum length allowed for that type of command, the system truncates the command.

Communication Between the Exits

Common Data Area: The MVS commands exits receive from the system the address of a 12-byte common data area in the exit routine parameter list. The common data area allows the exit routines to:

- Share data (in common work areas) across invocations.
- Supply the address of an installation-supplied routine that will clean up the common work areas when the exit routines are deleted. Deleted exit routines occur when:
  - MPF terminates (via a SET MPF=NO command) or
  - MPF is refreshed with a new MPFLSTxx that contains a new .CMD userexit name. MPF is then refreshed with that member using the SET MPF=xx command.

Sharing data: To enable your MVS commands exit routines to share data across invocations, code one of the exit routines to:

1. Create work areas in the extended common storage area (ECSA) by issuing a GETMAIN or STORAGE macro.
2. Place the addresses of the work areas in the second and third words of the common data area.

Whenever they are invoked, the MVS commands exit routines can access the common data area to obtain the addresses of the work areas. If the data required by the exits is 8 bytes or less, you can place the data itself within the second and third words of the common data area instead of creating work areas.

Supplying the address of a clean-up routine: You can also use the common data area to hold the address of an installation-supplied clean-up routine that will get control when the exits are to be deleted. The clean-up routine can perform any processing that is usually associated with exit routine work areas (such as releasing storage or clearing a control block). When you want to use a common data area specified clean-up routine, do the following:

1. Code the clean-up routine.
2. Supply the address of the clean-up routine in the first word of the common data area, which is pointed to by CMDXCWKP (the system initializes this field to 0).
3. The clean-up routine will be invoked when:

a. MPF terminates or
b. MPF is refreshed with a new MPFLSTxx that contains a new .CMD userexit name.

4. The clean-up routine will get control via a BALR 14, 15 instruction with the CMDXCCDA bit set to 1. Register 1 will be a pointer to an address that points to CMDX.

Note: There is a restriction on the data that resides in the first word of a storage pointed to by CMDXCWKP. It MUST be a valid address of your clean-up routine or zero. During command exit termination, the SET MPF processor checks the first word of the storage pointed to by CMDXCWKP, and if it is non-zero BALR 14, 15 is executed, where register 15 contains the first word pointed to by CMDXCWKP. So, if the first word pointed to by CMDXCWKP is not a valid address of a clean-up routine (it contains any non-zero value) an ABEND can be encountered.

Setting up the common data area: Normally, the first exit routine that anchors work areas from the last 8 bytes of the common data area will also initialize the first 4 bytes to the address of the clean-up routine, as shown in Table 6.

Table 6. Setting Up the Common Data Area

Field	Description
Word 1	Address of an installation-supplied clean-up routine that the exit routine calls when it is to be deleted
Word 2	Address of an installation-defined value (such as the address of a work area)
Word 3	Address of an installation-defined value (such as the address of a work area)

The system initializes the common data area to 0; thereafter, the common data area contains whatever values the exit routines place in it.

The exit routines must manage serialization of the common data area.

Individual Data Area

In addition to the common data area, the MVS commands exit routines each receive from the system the address of an individual 8-byte data area (in field CMDXIWKP of the CMDX) whenever they are invoked. Each exit routine can use its individual data area to:

- Pass data to itself (in a work area) across invocations
- Process the data during exit deletion

Passing data to itself: To enable an exit routine to pass data to itself across invocations, code the exit routine to:

1. Create a work area in the ECSA by issuing a GETMAIN or STORAGE macro
2. Place the address of the work area in the individual data area.

To obtain the address of the work area, code the exit routine to access the individual data area. As with the common data area, each individual data area is initialized to zero by the system and subsequently contains whatever values the exit routine places in it.

Processing during exit deletion: A command exit will be invoked once before deletion if its individual work area is non-zero. For this final invocation, the bit

CMDXCIDA will be set to 1. The exit should then clean up and free any storage pointed to from the individual work area. Exit deletion occurs when:

1. MPF terminates or
2. MPF is refreshed with a new MPFLSTxx that contains a new .CMD userexit name.

Note: There is NO restriction on the data pointed to by CMDXIWKP. The data can be bits or pointers to other data areas that contain data.

Each exit routine must manage serialization of its individual data area.

Macro Instructions and Restrictions

The MVS commands exit routines can issue system macros, but you should be aware of the following restrictions:

- Do not install an exit routine that issues the WAIT macro or calls a service that issues a WAIT. WAITs and implied WAITs can terminate console communications.
- Do not use macros whose expansions store data into an in-line parameter list.
- Do not issue the GETMAIN or STORAGE macro for subpools that represent space within a region (0 — 127, 240, or 250—252). Because the exit routines execute as part of the control program, they can use subpools 229, 230, and 249.
- Do not issue the DYNALLOC macro. Requesting dynamic allocation functions can cause an abend if your exit is processing a command that originated from a console. To avoid the abend, create a subsystem that runs in its own address space, and request dynamic allocation functions through that subsystem.

Security Consideration

It is the responsibility of your installation to provide any required security for an exit routine that issues system commands. For example, the routine can issue the RACROUTE REQUEST=VERIFYX macro to obtain the user token for a user id that is authorized to the command and then append the security token to the MGCRE macro parameter list. See System Authorization Facility (SAF) in the *z/OS MVS Programming: Authorized Assembler Services Guide* for further information on the security interface.

Entry Specifications

On entry, register 1 points to the address of the MVS commands exit parameter list (CMDX).

Registers at Entry: The contents of the registers on entry to the exit are as follows:

Register	**Contents**
0	Not applicable
1	Address of the pointer to the CMDX
2-12	Not applicable
13	Register save area
14	Return address
15	Entry point address of the exit routine

Parameter List Contents: Register 1 contains a pointer to the address of the commands exit parameter list (CMDX). The CMDX is mapped by the IEZVX101 macro (data area CMDX). See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for a description of the IEZVX101 mapping.

Return Specifications

Registers at Exit: Upon return from the exit, the register contents must be:

Register	Contents
0-14	Restored to contents at entry
15	One of the following return codes:

Return Code	Explanation
0	Indicates that the exit routine requests that the system process the command in the parameter list. The routine takes no action in processing the command itself.
2	Indicates that the user is not authorized to issue the command. No more exits are invoked for the command, and the subsystem call is bypassed.
4	Indicates that the exit routine (or module called from the routine) has processed the command. On return, the system takes no further action.
8	Indicates that the exit (or module called by the exit) should process the command but cannot at this time. The system issues message IEE707I. No more exits are invoked, and the subsystem call is bypassed.

Any value higher than 8 indicates an error condition.

Note: Installation-modified command text is used only when the exit routine returns a 0 return code.

Coded Example of the Exit Routine

The following is a coded example of an MVS commands installation routine that can be used to modify command processing:

```
*********************************************************************
*                                                                   *
* MODULE NAME      : CMDXIT                                         *
*                                                                   *
* DESCRIPTIVE NAME : SAMPLE COMMUNICATIONS TASK INSTALLATION EXIT *
*                    MODIFYING DISPLAY COMMANDS.                    *
*                                                                   *
* FUNCTION         : FOR DISPLAY TIME COMMANDS,                     *
*                    THIS EXIT PERMITS THE COMMAND TO BE            *
*                    ENTERED WITHOUT THE BLANK BETWEEN THE VERB     *
*                    AND THE PARAMETER. SPECIFICALLY                *
*                    - DT BECOMES DISPLAY T                         *
*                                                                   *
*  OPERATION       : GET THE ADDRESS OF THE COMMAND BUFFER WHICH    *
*                      IS POINTED TO BY CMDXCLIP.                   *
*                    DETERMINE IF THE COMMAND IS 'DT'. NO           *
```

```
*                    PROCESSING IS DONE IN THIS EXIT FOR ANY     *
*                    OTHER COMMANDS.                             *
*                   FOR DT CHANGE THE COMMAND TO DISPLAY T.      *
*                                                                *
*  NOTES           : FIRST 2 BYTES OF THE COMMAND BUFFER         *
*                    CONTAINS THE LENGTH OF THE COMMAND AND      *
*                    THE REST OF THE BUFFER CONTAINS THE COMMAND  *
*                    IMAGE. THE BUFFER IS 128 BYTES LONG.        *
*                                                                *
* ENTRY POINT      : CMDXIT                                      *
*                                                                *
*  PURPOSE         : ADD A BLANK BETWEEN THE DISPLAY VERB AND    *
*                    SUBPARAMETER 'T' TO PERMIT A SHORTER FORM   *
*                    OF THE COMMAND.                             *
*                                                                *
*  LINKAGE         : BALR                                        *
*                                                                *
*  INPUT DATA      : REG1 POINTER TO THE ADDRESS OF THE CMDX     *
*                  : REG13 ADDRESS OF STANDARD SAVE AREA         *
*                  : REG14 RETURN ADDRESS                        *
*                  : REG15 ENTRY POINT                           *
*                                                                *
*  REGISTERS SAVED : REG14 - REG12                               *
*                                                                *
*  REGISTER USAGE  : REG0 - PARAMETER REGISTER                   *
*                    REG1 - PARAMETER REGISTER                   *
*                    REG2 - WORK REGISTER                        *
*                    REG3 - WORK REGISTER                        *
*                    REG4 - WORK REGISTER                        *
*                    REG5 - CMDX                                 *
*                    REG6 - COMMAND BUFFER                       *
*                    REG7 - UNUSED                               *
*                    REG8 - UNUSED                               *
*                    REG9 - UNUSED                               *
*                    REG10 - UNUSED                              *
*                    REG11 - UNUSED                              *
*                    REG12 - MODULE BASE REGISTER                *
*                    REG13 - STANDARD SAVE AREA                  *
*                    REG14 - RETURN ADDRESS                      *
*                    REG15 - RETURN CODE ON EXIT                 *
*                                                                *
*  REGISTERS RESTORED :  REG14 - REG12                           *
*                                                                *
* CONTROL BLOCKS   :                                             *
*                                                                *
*  NAME        MAPPING MACRO      REASON USED             USAGE  *
*  ----        -------------      -----------             -----  *
*  CMDX        IEZVX101           CMD INSTALLATION EXIT   R,W    *
*                                 PARAMETER LIST                 *
*                                                                *
*  KEY = R-READ, W-WRITE, C-CREATE, D-DELETE                     *
*                                                                *
*  DATA TABLE       : NONE                                       *
*                                                                *
*  DATA AREA        : NONE                                       *
*                                                                *
* EXECUTABLE MACROS : WTO                                        *
*                                                                *
*                                                                *
******************************************************************
CMDXIT   CSECT
CMDXIT   AMODE 31                    31 BIT ADDRESSING MODE
CMDXIT   RMODE ANY                   31 BIT RESIDENCE
REG1     EQU   1
REG2     EQU   2
REG4     EQU   4
CMDXPTR  EQU   5
```

```
BUFFPTR  EQU   6
REG12    EQU   12
REG14    EQU   14
REG15    EQU   15
         EJECT ,
         BAKR  REG14,0                SAVE CALLER'S REGISTERS
         BALR  REG12,0                ESTABLISH MODULE BASE
         USING *,REG12
         L     CMDXPTR,0(REG1)        GET CMDX ADDRESS
         USING CMDX,CMDXPTR           ACCESS THE CMDX
         L     BUFFPTR,CMDXCLIP       GET THE COMMAND BUFFER ADDRESS
         USING CMDXCLIB,BUFFPTR       ACCESS THE BUFFER
         LA    REG2,CMDXCMDI          ACCESS START OF TEXT
         CLC   0(L'DT,REG2),DT        IS THIS DT
         BNE   EXIT                   NO, NO PROCESSING FOR COMMAND
*********************************************************************
*        PROCESS THE DT COMMAND.                                    *
*        1. ALTER THE TEXT IN THE COMMAND BUFFER TO D T             *
*        2. INDICATE TEXT CHANGE REQUEST                            *
*********************************************************************
         MVC   CMDXCMDI(L'D_T),D_T    MOVE IN D T
         LA    REG4,L'D_T             GET NEW COMMAND LENGTH
         STH   REG4,CMDXCMDL          STORE NEW COMMAND LENGTH
         OI    CMDXRFL1,CMDXRCMI      REQUEST TEXT CHANGE
EXIT     EQU   *
         XR    REG15,REG15            SYSTEM TO PROCESS COMMAND
         PR                           RETURN TO CALLER
DT       DC    C'DT'                  SHORT FORM OF D T
D_T      DC    C'D T'                 REAL D T
         IEZVX101
         END   CMDXIT
```

Chapter 45. MVS Message Service (MMS) Exits

Topics for This Exit Appear as Follows:

The MVS message service (MMS) enables you to translate U.S. English messages into other languages.

If you are routing system messages to a TSO/E extended MCS console, TSO/E will display translated messages in the primary language associated with the TSO/E session. If MMS is active, users of extended MCS consoles on TSO/E can select available languages for message translation and the system can display translated messages on the user's screen. TSO/E terminal users can also receive on their terminals translated TSO/E messages and the translated messages of any application that directs its messages to TSO/E and uses MMS services directly or through TSO/E services. To receive translated messages on TSO/E terminals, you must have TSO/E Version 2.2 installed on your system.

MMS provides two installation exit points that allow you to modify MMS processing. The MMS exits are invoked when application programs issue macros to request the following user functions:

- Message translation
- Language query (to see if a particular language is available).

For information on MMS and MMS user functions, see *z/OS MVS Programming: Assembler Services Guide*.

For information on MMS macros, see *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

There are two MMS exits: the MMS preprocessing exit and the MMS postprocessing exit. The MMS preprocessing exit is invoked before MMS processes the user function. The MMS postprocessing exit is invoked after MMS processes the user function.

The MMS preprocessing exit can be used to:

- Replace existing translations of selected messages with other installation-defined translations.
- Prevent translation of selected messages.
- Set a local communication word that will be passed to the MMS postprocessing exit. Use this field to pass any information to your postprocessing exit.

The MMS postprocessing exit can be used to:

- Collect MMS usage statistics.

Installing the Exit Routines

The MMS exits must reside in an APF-authorized load library included in the LNKLST concatenation.

Statements that identify either an MMS preprocessing exit routine, an MMS postprocessing exit routine, or both, must be specified in the appropriate MMSLSTxx member of SYS1.PARMLIB.

In MMSLSTxx, do the following on an EXIT statement:

- On the NUMBER keyword:
  - To identify an MMS preprocessing exit, specify: (1)
  - To identify an MMS postprocessing exit, specify: (2).
- Specify the module name of the exit routine on the ROUTINE parameter.

For example, to identify both an MMS preprocessing exit (MMSEXIT1) and an MMS postprocessing exit (MMSEXIT2) in an MMSLSTxx parmlib member, specify:

```
EXIT NUMBER(1)  ROUTINE(mmsexit1)
EXIT NUMBER(2)  ROUTINE(mmsexit2)
```

No more than two MMS exit routines can be specified in an MMSLSTxx parmlib member.

For more information on the MMSLSTxx parmlib member, see *z/OS MVS Initialization and Tuning Reference*.

For general instructions on installing an exit routine, see "Link editing an Installation Exit Routine into a Library" on page 3.

Replacing the Exit Routines: To replace an MMS exit routine when MMS is active, you must either:

- Modify the MMSLSTxx parmlib member to name a different exit routine on the ROUTINE parameter and issue the SET MMS=xx command to refresh the parmlib member.
- Issue the SET MMS=xx command for the MMSLSTxx parmlib member that names the required MMS exit routine.

For more information on the SET MMS command, see *z/OS MVS System Commands*.

Exit Routine Environment

The exit routines receive control in the following environment:

- Enabled for interrupts.
- In supervisor state with PSW key 0.
- In AMODE 31 and RMODE ANY.
- In the MMS address space.
- In pageable storage.
- In cross memory mode.

Exit Recovery: MMS provides a functional recovery routine (FRR) to protect the exits.

If the MMS exit routine abends, the system will shut down MMS. The system recovery routine will request an SDUMP that can be used by the installation to debug the exit routine.

Exit Routine Processing

The MMS preprocessing exit routine, if one is defined, is invoked **before** MMS processes the input parameter block (and its related data areas). The MMS postprocessing exit is invoked immediately **after** MMS processes the input parameter block but before the block has been copied back to the caller's address space.

The system passes the address of the input parameter block to the exits in word 1 of the exit routine parameter list (pointed to by register 1). For message translation, the input parameter block is a message input/output block (MIO), which is mapped by the CNLMMIO macro. For a language query, the input parameter block is a language query block (LQB), which is mapped by the CNLMLQB macro. The exit routines can modify these blocks as needed.

If your installation defines MMS exits, MMS will invoke the exits for both the message translation and language query functions. The system indicates the type of function for which the exit has been invoked in register 0 with one of the following codes:

Code	Meaning
1	Message translation
4	Language query

Message Translation: The MMS exits are invoked each time an application invokes MMS with a request to translate a message (register 0 is set to function code 1). If you code MMS exit routines to modify MMS translation, you will probably modify translations of ***selected*** messages. For the ***majority*** of MMS translations, the exit routines will allow the translations to occur without modification (as if the exit routines did not exist).

Allowing MMS Translations to Occur: To allow MMS translation of a message to occur, the exit routines simply return control to the system:

- Without resetting the default of zero that is in the exit processing indicator (pointed to in word 3 of the parameter list)
- With a zero return code in register 15.

The installation application receives the MMS translated message text.

Modifying MMS Translations: When you want to modify MMS translations, use the preprocessing exit routine. The MMS preprocessing exit is invoked **before** MMS translates a message. The exit routine can examine the input message text (pointed to by the MIO control block) and do one of the following:

- Replace the message with installation-preferred message text
- Prevent the message from being translated.

Using the Preprocessing Exit: To replace or prevent MMS translations of selected messages, code the preprocessing exit routine to set the exit processing indicator to a nonzero value and do one of the following:

- Place an installation-defined message in the message output area (also pointed to in the MIO) and return control to the system. The installation-defined message is returned to the application that invoked MMS.
- Return control to the system, thus preventing translation. The original, U.S. English form of the message is returned to the application that invoked MMS.
- Set register 15 to a nonzero value and return control to MMS. The translation request will be terminated.

The exit routine must return a zero return code in register 15. Otherwise, MMS will terminate the translation request.

With one exception, the postprocessing exit routine (if one is defined) will be invoked even when MMS translation is bypassed. The exception is when the preprocessing exit routine indicates an error (by placing a value greater than 8 in a fullword pointed to by word 4 of the exit parameter list). See “Return Specifications” on page 330 (Using Words 4 and 5) for more information.

Using the Postprocessing Exit: The MMS postprocessing exit (if one is defined) is invoked ***after*** MMS has translated a message, but before the message is returned to the end user.

Like the MMS preprocessing exit routine, the MMS postprocessing exit routine can examine the input message parameters and place installation-preferred message text in the message output area. Because it is invoked after MMS message translation has occurred, the MMS postprocessing exit routine cannot bypass MMS processing. However, the postprocessing exit routine can do one of the following:

- Gather statistics
- Set the exit processing indicator to a nonzero value and return control to MMS. The original, U.S. English form of the message is returned to the application.

Language Query: The MMS exits are invoked when MMS receives a query to determine which languages are available for message translation (register 0 is set to function code 4).

When invoked during a language query, an MMS exit could:

- Collect usage statistics
- Modify the list of languages to be returned to the application (in the LQB).

Programming Considerations

Using Information in the Exit Parameter List:

- The system passes the address of the input parameter block to the exits in word 1 of the exit routine parameter list (pointed to by register 1). For message translation, the input parameter block is a message input/output block (MIO). For a language query, the input parameter block is a language query block (LQB). The exit routines can modify these blocks as needed. For the mappings of the MIO and LQB control blocks, see *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.
- The system passes to the exit routines the address of a local communication word in word 2 of the exit parameter list. The pre and postprocessing exit routines can use the local communication word to share data (or the address of data) for one user invocation of MMS.
- The system provides the address of the exit processing indicator in word 3 of the exit parameter list. An MMS exit routine can cause MMS processing of a particular message to be bypassed by setting the exit processing indicator to a nonzero value. The exit processing indicator is set to 0 each time MMS invokes the exit.
- The exit routines can indicate processing errors by placing installation-defined return and reason codes in areas pointed to by words 4 and 5, respectively, of the exit routine parameter list. See "Return Specifications" on page 330 (Using Words 4 and 5) for more information.
- The system passes to the exit routines the address of a 512-byte work area in word 6 of the exit routine parameter list. The exits can use the work area for the current invocation of the service. Additional storage, if required, must be obtained by the exit routines by issuing a STORAGE or branch-entry GETMAIN macro.

Other Considerations:

- Code the exit routines to be reentrant.
- Because the exit routines run in cross memory mode, they cannot issue SVCs.
- The exit routines are invoked each time MMS processes the message translate or language query functions. Therefore, when coding the exit routines, you should be aware that an increased path length will increase processor use and affect performance.
- The exit routines must follow standard linkage conventions.

Macro Instructions and Restrictions: The exit routines can only use services that run in cross-memory mode. See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information on services that run in cross-memory mode.

Entry Specifications

MMS passes to the exit the address of the input parameter block (either an MIO or an LQB, depending on the type of request that MMS has been called to process).

Registers at Entry: The contents of the registers on entry to the MMS exit routines are:

Register	Contents
0	Type of MMS function the installation application requested (1 for message translation, 4 for language query)

1 Address of the exit parameter list

2-12 Undefined

13 Address of an 18-fullword register save area

14 Return address

15 Entry point address of exit

Parameter Descriptions: Register 1 points to the following list of addresses:

Word 1
The address of an MIO or LQB control block.
- When the exit is invoked for the message translation function, this field will point to the address of an MIO (mapped by macro CNLMMIO).
- When the exit is invoked for the language query function, this field will point to the address of an LQB (mapped by macro CNLMLQB).

Word 2
The address of a local communication word.

Word 3
The address of the exit processing indicator. The indicator is set to 0 by default.

Word 4
The address of an optional, installation-defined return code.

Word 5
The address of an optional, installation-defined reason code.

Word 6
The address of a 512-byte storage area (starting on a double-word boundary) that the exit routines can use for the current invocation of MMS.

Return Specifications

The exit routines return control to the system with:
- A value in the exit processing indicator, pointed to by Word 3 of the exit parameter list.
- Optional, installation-defined return and reason codes, pointed to by words 4 and 5 of the exit parameter list.
- A return code in register 15.

Preprocessing Exit Routine: The exit routine indicates whether to bypass the requested MMS function by setting the exit processing indicator to a nonzero value. If the exit processing indicator is not set, the system will process the function.

Using Words 4 and 5: The preprocessing exit routine can return optional, installation-defined return and reason codes in areas pointed to by words 4 and 5, respectively, of the exit parameter list. When the exit routine sets the exit processing indicator to a nonzero value, the system checks the value in the area pointed to by word 4. The exit routine can indicate that it has encountered a serious error by setting the field pointed to by word 4 to a value greater than 8. When this happens, MMS terminates the user request. Otherwise, if the field pointed to by word 4 is set to a value of 8 or less, or the exit routine does not set

the exit processing indicator, MMS continues processing and the fields pointed to by words 4 and 5 are passed as information to the MMS postprocessing exit routine (if one is defined).

Postprocessing Exit Routine: Like the preprocessing exit routine, the postprocessing exit routine can cause MMS processing to be bypassed by setting the exit processing indicator to a nonzero value and returning control to the system with a zero return code in register 15. However, any values the exit routine places in the fields pointed to by words 4 and 5 of the exit parameter list are not used by the system.

Registers at Exit: Upon return from exit routine processing, the register contents must be:

Register
Contents

0,1 Undefined

2-14 Restored to contents at entry

15 One of the following return codes:

Return Code
Explanation

0 The exit routine has completed processing.

nonzero
The exit routine has encountered an error. MMS will terminate the user request.

Coded Examples of MMS Exit Routines

This topic contains the following sample MMS exit routines:

- MMSEXIT1. This MMS preprocessing exit routine prevents translation of a particular language.
- MMSEXIT2. This MMS postprocessing exit routine collects MMS usage statistics.

MMSEXIT1 -- Preventing Translations of a Particular Language: In the following example, an installation uses the MMS preprocessing exit routine to prevent MMS from translating messages into Japanese.

The installation places the language code of the language for which translations are to be suppressed (in this case, JPN is the language code) into the CVTUSER field of the CVT so that the exit routine can access it.

When the installation's exit routine is invoked for a translation request, the routine compares the language code in the CVTUSER field with the language code contained in the MIO. When the request is for Japanese translation (MIO contains 'JPN'), the exit routine causes MMS translation to be bypassed by setting the exit processing indicator (pointed to by word 3 of the exit parameter list) to a nonzero value and returning control.

```
MMSEXIT1 CSECT
MMSEXIT1 AMODE  31
MMSEXIT1 RMODE  ANY
         STM    14,12,12(13)
         BALR   12,0
         USING  *,12
*********************************************************************
```

```
***                                                                   *
***     FOR A TRANSLATION REQUEST, THIS EXIT WILL COMPARE THE         *
***     LANGUAGE CODE CONTAINED IN THE CVTUSER FIELD OF THE CVT       *
***     WITH THAT CONTAINED IN THE MIO FOR THIS REQUEST. THIS EXIT    *
***     ASSUMES THE INSTALLATION HAS UPDATED THE CVTUSER FIELD TO     *
***     CONTAIN THE DESIRED THREE CHARACTER LANGUAGE CODE. IF THE     *
***     LANGUAGE CODE CONTAINED IN THE CVTUSER FIELD MATCHES THE      *
***     CODE IN THE MIO, THE TRANSLATION REQUEST WILL BE              *
***     TERMINATED BY SETTING THE EXIT PROCESSING INDICATOR           *
***     TO A NON-ZERO VALUE.                                          *
***                                                                   *
***                                                                   *
*************************************************************************
*
         C     R0,ONE                            TRANSLATION REQUEST?
         BNE   END                               NO, END PROCESSING
         L     R3,0(,R1)                         OBTAIN MIO ADDRESS
         L     R2,16(0,R0)                       OBTAIN CVT ADDRESS
         CLC   CVTUSER-CVT(3,R2),MIOLANG-MIO(R3) COMPARE LANG. CODES
         BNE   END                               EXIT IF NOT EQUAL
         L     R3,8(,R1)                         OBTAIN PROCESSING    C
                                                   INDICATOR ADDRESS  C
         MVC   0(4,R3),ONE                       SET PROCESS          C
                                                  INDICATOR TO        C
                                                  PREVENT TRANSLATION C
END      DS    0H
         LM    14,12,12(13)
         SLR   15,15
         BR    14
*************************************************************************
ONE      DC    F'1'
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
*************************************************************************
         DSECT
         CVT   DSECT=YES
         CNLMMIO
         END MMSEXIT1
```

MMSEXIT2 -- Collecting MMS Usage Statistics: In the following example, an installation uses the MMS postprocessing exit routine to track the number of successful and unsuccessful (successful=RC0, unsuccessful=nonzero) translations of U.S. English into Japanese.

The installation first creates a work area to contain counters of successful and unsuccessful translations. The routine places the address of the work area in the CVTUSER field of the CVT so that the postprocessing exit routine can find the work area and update the counters.

The installation can use the following MMS postprocessing exit routine to determine whether the translation was successful. Based on the result, the routine updates the appropriate counter in the work area (pointed to in CVTUSER).

```
MMSEXIT2 CSECT
MMSEXIT2 AMODE  31
MMSEXIT2 RMODE  ANY
         STM    14,12,12(13)
         BALR   12,0
         USING  *,12
*************************************************************************
***     FOR A TRANSLATION REQUEST, THIS EXIT ROUTINE WILL OBTAIN      *
***     THE ADDRESS OF AN INSTALLATION DEFINED CONTROL BLOCK          *
```

```
***     (POINTED TO BY THE CVTUSER FIELD OF THE CVT CONTROL BLOCK).   *
***     IF THE INSTALLATION CONTROL BLOCK EXISTS,                     *
***     A CHECK OF THE SUCCESS OR FAILURE OF THE TRANSLATION          *
***     REQUEST WILL BE MADE AND THE APPROPRIATE SUCCESS/FAILURE      *
***     COUNTER IN THE INSTALLATION CONTROL BLOCK WILL BE UPDATED.    *
*********************************************************************
         C     R0,ONE                       TRANSLATION REQUEST?
         BNE   END                          NO, END PROCESSING
         L     R2,0(,R1)                    OBTAIN MIO ADDRESS
         CLC   MIOLANG-MIO(3,R2),=C'JPN'    COMPARE LANGUAGE           C
                                              CODES
         BNE   END                          EXIT IF NOT EQUAL
         L     R2,0(,R0)
         L     R3,CVTUSER-CVT(,R2)          OBTAIN INSTALLATION        C
                                              CONTROL BLOCK ANCHOR
         LTR   R3,R3                        INSTALLATION CONTROL BLOCK C
                                              PRESENT?
         BZ    END                          NO,END
         L     R4,12(,R1)                   OBTAIN RETURN CODE ADDRESS
         L     R4,0(,R4)                    OBTAIN RETURN CODE
         LTR   R4,R4                        TEST RETURN CODE
         BNZ   FAIL
         L     R5,INSTLSUC-INSTLCB(R3)      OBTAIN SUCCESS COUNTER
         LA    R5,1(,R5)                    BUMP COUNTER
         ST    R5,INSTLSUC-INSTLCB(R3)      SAVE COUNTER
         B     END                          GOTO END
FAIL     L     R5,INSTLFAL-INSTLCB(R3)      OBTAIN FAILURE COUNTER
         LA    R5,1(,R5)                    BUMP COUNTER
         ST    R5,INSTLFAL-INSTLCB(R3)      SAVE COUNTER
END      DS    0H                           RETURN
         LM    14,12,12(13)
         SLR   15,15
         BR    14
*********************************************************************
ONE      DC    F'1'
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
*********************************************************************
         DSECT
         CVT   DSECT=YES
         CNLMMIO
INSTLCB  DSECT               INSTALLATION CONTROL BLOCK
INSTLACR DS    CL4'INST'     INSTALLATION CONTROL BLOCK ACRONYM
INSTLSUC DS    F             LANGUAGE USAGE COUNTER FOR JAPANESE       C
                               SUCCESSFUL TRANSLATIONS
INSTLFAL DS    F             LANGUAGE USAGE COUNTER FOR JAPANESE       C
                               UNSUCCESSFUL TRANSLATIONS
         DS    CL12          RESERVED
INSTLLEN EQU   *-INSTLCB
         END MMSEXIT2
```

Part 3. Installation Exit Directory

This section contains a list of installation exits that are coded into the various components of MVS and DFP and related program products. Each installation exit has an entry that lists its name and a short description or title that explains its use.

The installation exits listed in this section are described in other publications. The exits are listed by the title and order number of the manual that contains the documentation for the exit.

This directory contains entries for the following components and program products:

Chapter 46. BCP Exits

z/OS MVS System Management Facilities (SMF) describes the exits listed in Table 7.

Table 7. BCP Exits (USERx)

Exit	Description
USER1(name)	Exit routine is given control after each record is read by SMF dump program
USER2(name)	Exit routine is given control only when the dump program selects a record to be written
USER3(name)	Exit routine is given control after the output data set is closed by SMF dump program

z/OS MVS Programming: Authorized Assembler Services Guide describes the exit listed in Table 8.

Table 8. BCP Exit (authorized assembler)

Exit	Description
IEAVTRML	Allows installation to supply resource management routines

z/OS MVS Programming: JES Common Coupling Services describes the exits listed in Table 9.

Table 9. BCP Exits (JES common coupling services)

Exit	Description
IXZXIT01	Allows installation to view, modify, or reroute a message or acknowledgement before the message arrives at the receiving member's mailbox.
IXZXIT02	Allows installation to view or modify a message before it is retrieved from a mailbox.
IXZXIT03	Allows installation to attach to or detach from an installation-defined JES XCF group.

Chapter 47. DFSMS Exits

Table 10 lists the DFSMS *installation exits*, described in *z/OS DFSMS Installation Exits* and *z/OS DFSMS OAM Planning, Installation, and Storage Administration Guide for Tape Libraries*.

The list in Table 10 does not include DFSMS user exits.

Table 10. DFSMS Exits

Exit	Description
CBRHADUX	Object access method (OAM) auto-delete
CBRUXCUA	Change use attribute installation exit
CBRUXEJC	Cartridge eject installation exit
CBRUXENT	Cartridge entry installation exit
IDAEOVXT	VSAM EOV installation exit
IEFXVNSL	Automatic volume recognition (AVR) nonstandard label processing
IFG0EX0A	Format-1 DSCB not found during OPEN or EOV
IFG0EX0B	Take control during OPEN for a DCB
IFG01991	Open, close, end of volume abnormal conditions
IFG0193G	ISO/ANSI/FIPS Version 3 label exits for volume access, file access, label validation, and label validation suppression
IGBDCSX1 IGBDCSX2	DASD precalculation and postcalculation services
IGDACSDC	Automatic class selection (ACS) data class exit
IGDACSSC	Automatic class selection (ACS) storage class exit
IGDACSMC	Automatic class selection (ACS) management class exit
IGG026DU	Catalog pre-initialization
IGG029DM	Process after DADSM SCRATCH failure
IGG029DU	DADSM SCRATCH pre-initialization
IGGDASU3	DADSM SCRATCH postprocessing
IGG030DU	DADSM RENAME pre-initialization
IGGDARU3	DADSM RENAME postprocessing
IGGPRE00 IGGPOST0	System provided DADSM pre-processing and post-processing exit routines, associated with the IGGPRE00_EXIT and IGGPOST0_EXIT dynamic exits, for allocate, extend, scratch, partial release and rename functions.
IGXMSGEX	Customize messages
NSLETRLI	Nonstandard label processing for input trailers
NSLETRLO NSLCTRLO	Nonstandard label processing for output trailers
NSLOHDRI NSLEHDRI	Nonstandard label processing for input headers
NSLOHDRO NSLEHDRO	Nonstandard label processing for output headers

Table 10. DFSMS Exits (continued)

Exit	Description
NSLREPOS	Volume verification using the dynamic device reconfiguration (DDR) option for nonstandard label processing
NSLRHDRI	Nonstandard label processing for restarting after a checkpoint
OMODVOL1 EMODVOL1	Volume label editor for open and EOV

Chapter 48. IPCS Exits

z/OS MVS IPCS Customization describes the exits listed in Table 11.

Table 11. IPCS Exits

Exit	Description
ANALYZE exit	Generate data for contention analysis
ASCB exit	Generate information related to the address space or ASCB being processed
BLSUGWDM validity check routine	Command validation routine for IPCS
Control block formatter exit	Assist in formatting a control block
Control block status (CBSTAT) exit	Perform analysis and generate condensed output describing information relevant to the debugging process
CTRACE buffer find exit	Locate the component trace buffers in a dump for a particular component
CTRACE filter/analysis (CTRF) exit	• Perform statistical analysis of the component trace • Provide additional component trace filtering • Limit the number of component trace entries processed
Find exit	Associate a symbol with an AREA or STRUCTURE in a dump
GTFTRACE filter/analysis exit	• Do statistical analysis of GTF trace records • Provide additional GTF trace record filtering • Limit the number of GTF trace records processed
GTFTRACE formatting appendage	Format GTF trace records containing a particular FID and EID format in the user range.
Model processor formatting (MPF) exit	Dynamically interact with the formatting service to augment its function
Post-formatting exit	Supply a routine for any type of structure that can be described by a parmlib data statement
Scan exit	Check the validity of an AREA or STRUCTURE in a dump
Task control block (TCB) exit	Generate information related to the task control block (TCB) being processed
Verb exit	Format and print dump data sets created by stand-alone, SVC or SYSMDUMP dumping services

Chapter 49. JES2 Exits

z/OS JES2 Installation Exits describes the exits listed in Table 12.

Table 12. JES2 Exits

Exit	Description
Exit 0	Pre-initialization
Exit 1	Print/Punch Separators
Exit 2	JOB Statement Scan
Exit 3	JOB Statement Accounting Field Scan
Exit 4	JCL and JES2 Control Statement Scan
Exit 5	JES2 Command Preprocessor
Exit 6	Converter/Interpreter Text Scan
Exit 7	JCT Read/Write (JES2)
Exit 8	Control Block Read/Write (USER)
Exit 9	Job Output Overflow
Exit 10	$WTO Screen
Exit 11	Spool Partitioning Allocation ($TRACK)
Exit 12	Spool Partitioning Allocation ($STRAK)
Exit 13	TSO/E Interactive Data Transmission Facility Screening and Notification
Exit 14	Job Queue Work Select — $QGET
Exit 15	Output Data Set/Copy Select
Exit 16	Notify
Exit 17	BSC RJE SIGNON/SIGNOFF
Exit 18	SNA RJE LOGON/LOGOFF
Exit 19	Initialization Statement
Exit 20	End of Input
Exit 21	SMF Record
Exit 22	Cancel/Status
Exit 23	FSS Job Separator Page (JSPA) Processing
Exit 24	Post-initialization
Exit 25	JCT Read (FSS)
Exit 26	Termination/Resource Release
Exit 27	PCE Attach/Detach
Exit 28	Subsystem Interface (SSI) Job Termination
Exit 29	Subsystem Interface (SSI) End-of-Memory
Exit 30	Subsystem Interface (SSI) Data Set OPEN and RESTART
Exit 31	Subsystem Interface (SSI) Allocation
Exit 32	Subsystem Interface (SSI) Job Selection
Exit 33	Subsystem Interface (SSI) Data Set CLOSE
Exit 34	Subsystem Interface (SSI) Data Set Unallocation

Table 12. JES2 Exits (continued)

Exit	Description
Exit 35	Subsystem Interface (SSI) End-of-Task
Exit 36	Pre-security Authorization Call
Exit 37	Post-security Authorization Call
Exit 38	TSO/E Receive Data Set Disposition
Exit 39	NJE SYSOUT Reception Data Set Disposition
Exit 40	Modifying SYSOUT Characteristics
Exit 41	Modifying Output Grouping Key Selection
Exit 42	Modifying a Notify User Message
Exit 43	Transaction Program Select/Terminate/Change
Exit 44	JES2 Converter Exit (Main Task)
Exit 45	Pre-SJF Exit Request
Exit 46	Transmitting an NJE Data Area
Exit 47	Receiving an NJE Data Area
Exit 48	Subsystem Interface (SSI) SYSOUT Data Set Unallocation
Exit 49	Job Queue Work Select — QGOT
Exit 50	End of Input
Exit 51	Job Phase Change Exit ($QMOD)
Exit 52	JOB JCL Statement Scan (JES2 User Environment)
Exit 53	JOB Statement Accounting Field Scan (JES2 User Environment)
Exit 54	JCL and JES2 Control Statement Scan (JES2 User Environment)
Exit 55	NJE SYSOUT Reception Data Set Disposition
Exit 56	Modifying an NJE Data Area before Its Transmission
Exit 57	Modifying an NJE Data Area before Receiving the Rest of the NJE Job

Chapter 50. JES3 Exits

z/OS JES3 Customization describes the exits listed in Table 13.

Table 13. JES3 Exits

Exit	Description
IATUX03	Examine or modify converter/interpreter text created from JCL
IATUX04	Examine the job information from the JCL
IATUX05	Examine the step information from the JCL
IATUX06	Examine DD statement information from the JCL
IATUX07	Examine or substitute unit, type and volume serial information
IATUX08	Examine setup information
IATUX09	Examine final job status, JST and JVT
IATUX10	Generate a message
IATUX11	Inhibit printing of the LOCATE request or response
IATUX14	Job validation/restart LOCATE request or response
IATUX15	Scan an initialization statement
IATUX17	Define set of scheduler elements
IATUX18	Check input authority level for consoles
IATUX19	Examine or modify temporary OSE
IATUX20	Examine or modify data written on job header pages
IATUX21	Create and write data set headers for output data sets
IATUX22	Examine or alter the forms alignment
IATUX23	Examine or modify data written to trailer pages
IATUX24	Examine the Net-id and the devices requested
IATUX25	Examine or modify volume serial number
IATUX26	Examine MVS scheduler control blocks
IATUX27	Examine or alter the JDAB, JCT and JMR
IATUX28	Examine the accounting information as provided by the JOB statement
IATUX29	Examine the accounting information as provided by the JCT, JDAB and JMR
IATUX30	Examine authority level for TSO/E terminal commands
IATUX32	Override the DYNALDSN initialization statement
IATUX33	JES3 control statement and JCL EXEC statement installation exit
IATUX34	JCL DD statement user exit and JCL EXEC statement installation exit
IATUX35	Validity check network commands
IATUX36	Collect accounting information
IATUX37	Modify the JES3 networking data set header
IATUX38	Change SYSOUT class for networking data sets
IATUX39	Modify the data set header for a SYSOUT data set
IATUX40	Modify job header

Table 13. JES3 Exits (continued)

Exit	Description
IATUX41	Determines the disposition of job over JCL limit
IATUX42	TSO/E interactive data transmission facility screening and notification
IATUX43	Modify job header segments
IATUX44	Examine and modify the JCL
IATUX45	Examine and modify data sent to an output writer FSS
IATUX46	Select processors eligible for C/I processing
IATUX48	Override operator modification of output data sets
IATUX49	Override address space selected for C/I processing
IATUX50	Process user defined BSIDMOD codes for C/I processing
IATUX57	Select a single WTO routing code for JES3 MSGROUTE
IATUX58	Modify security information before JES3 security processing
IATUX59	Modify security information after JES3 security processing
IATUX60	Determine action to take when a TSO/E user is unable to receive a data set
IATUX61	During MDS processing, chooses whether a job should be canceled or sent to the error queue
IATUX62	Overrides the decision to accept a tape or disk mount
IATUX66	Assigns transmission priority to a SNA/NJE data stream
IATUX67	Determines action when remote data set is rejected by RACF
IATUX69	Determines if a message is to be sent to the JES3 global address space
IATUX70	Performs additional message processing

Chapter 51. RACF Exits

z/OS Security Server RACF System Programmer's Guide describes the exits listed in Table 14.

Table 14. RACF Exits

Exit	Description
ICHCCX00	Command exit
ICHCNX00	Command exit
ICHDEX01	RACF password authentication
ICHFRX01	RACROUTE request=FASTAUTH preprocessing
ICHFRX02	RACROUTE request=FASTAUTH postprocessing
ICHPWX01	New password exit
ICHRCX01	RACROUTE request=AUTH preprocessing
ICHRCX02	RACROUTE request=AUTH postprocessing
ICHRDX01	RACROUTE request=DEFINE preprocessing
ICHRDX02	RACROUTE request=DEFINE postprocessing
ICHRIX01	RACROUTE request=VERIFY preprocessing
ICHRIX02	RACROUTE request=VERIFY postprocessing
ICHRLX01	RACROUTE request=LIST pre/postprocessings
ICHRLX02	RACROUTE request=LIST selection
ICHRSMFE	RACF report writer
IRRACX01	ACEE compression/expansion
IRREVX01	Common command exit
ICHRFX03	RACROUTE request=FASTAUTH preprocessing
ICHRFX04	RACROUTE request=FASTAUTH postprocessing

z/OS Security Server RACF Callable Services describes the exit listed in Table 15.

Table 15. RACF Exit

Exit	Description
IRRSXT00	SAF callable services router

Chapter 52. RMF™ Exits

z/OS RMF User's Guide describes the exits listed in Table 16.

Table 16. RMF Exits

Exit	Description
ERBMFDUC	Internal processing
ERBMFEVT	User sampler
ERBMFIUC	Monitor I session initialization
ERBMFPUS	Post-processor
ERBMFRUR	Report writer
ERBMFTUR	Termination
ERBTRACE	Field tracing

Chapter 53. TSO/E Exits

z/OS TSO/E Customization describes the exits listed in Table 17.

Table 17. TSO/E Exits

Exit	Description
ADRS exit	Perform additional processing whenever a user selects the ADRS option from the Information Center Facility
REXX attention handling	Perform special attention processing
CHSFEXIT	Change default space parameters that VM/PC servers use to dynamically allocate MVS data sets
Edit exit for RENUM, MOVE, and COPY subcommands	Tailor the way line numbering is done
Edit exit for syntax checkers	Supply syntax checker with data set attributes
REXX exec initialization	Access or update REXX variables
REXX exec processing	Perform special processing before REXX exec executes
REXX exec termination	Access or update REXX variables
ICQAMFX1	Application manager function pre-initialization
ICQAMFX2	Application manager function post-termination
ICQAMPX1	Application manager panel pre-display
ICQAMPX2	Application manager panel post-display
IDYTSINI	TSOLIB initialization
IDYTSTER	TSOLIB termination
IKJADINI	ALTLIB initialization
IKJADTER	ALTLIB termination
IKJCNXCD	CONSPROF pre-display
IKJCNXCI	CONSPROF initialization
IKJCNXCT	CONSPROF termination
IKJCNXAC	CONSOLE activation
IKJCNXDE	CONSOLE deactivation
IKJCNX50	CONSOLE 80% message capacity
IKJCNX60	CONSOLE 100% message capacity
IKJCNXPP	CONSOLE pre-parse
IKJCT43I	EXEC initialization
IKJCT43T	EXEC termination
IKJCT44B	Add installation-written CLIST built-in functions
IKJCT44S	Add installation-written CLIST statements

Table 17. TSO/E Exits (continued)

Exit	Description
IKJEESXA	LISTBC failure
IKJEESXB	LISTBC termination
IKJEESX0	SEND initialization
IKJEESX1 IEEVSNX1	SEND pre-display
IKJEESX2 IEEVSNX2	SEND pre-save
IKJEESX3 IEEVSNX3	SEND failure
IKJEESX4 IEEVSNX4	SEND termination
IKJEESX5	LISTBC initialization
IKJEESX6	LISTBC pre-display
IKJEESX7	LISTBC pre-list
IKJEESX8	LISTBC pre-read
IKJEESX9	LISTBC pre-allocate
IKJEFD21	FREE initialization
IKJEFD22	FREE termination
IKJEFD47	ALLOCATE command initialization
IKJEFD49	ALLOCATE command termination
IKJEFF10	SUBMIT command
IKJEFF53	OUTPUT, STATUS and CANCEL commands
IKJEFLD	Logon pre-prompt
IKJEFLD1	Logon authorized pre-prompt
IKJEFLD2	Logoff
IKJEFLD3	Logon post-prompt
IKJEFLN1	Logon pre-display
IKJEFLN2	Logon post-display
IKJEFY11	OUTDES initialization
IKJEFY12	OUTDES termination
IKJEFY60	PRINTDS initialization
IKJEFY64	PRINTDS termination
IKJEGASI	TESTAUTH subcommand initialization
IKJEGAST	TESTAUTH subcommand termination
IKJEGAUI	TESTAUTH initialization
IKJEGAUT	TESTAUTH termination
IKJEGCIE	TEST subcommand initialization
IKJEGCTE	TEST subcommand termination
IKJEGMIE	TEST initialization
IKJEGMTE	TEST termination
IKJPRMX1	PARMLIB initialization
IKJPRMX2	PARMLIB termination
INMCZ21R	TRANSMIT/RECEIVE NAMES data set pre-allocation

Table 17. TSO/E Exits (continued)

Exit	Description
INMRZ01 INMRZ01R	RECEIVE initialization
INMRZ02 INMRZ02R	RECEIVE termination
INMRZ04 INMRZ04R	RECEIVE notification
INMRZ05R	RECEIVE acknowledgment notification
INMRZ06R	RECEIVE pre-acknowledgment notification
INMRZ11 INMRZ11R	RECEIVE data set preprocessing
INMRZ12 INMRZ12R	RECEIVE data set postprocessing
INMRZ13 INMRZ13R	RECEIVE data set decryption
INMRZ15R	RECEIVE post-prompt
INMRZ21R	RECEIVE log data set pre-allocation
INMXZ01 INMXZ01R	TRANSMIT startup
INMXZ02 INMXZ02R	TRANSMIT termination
INMXZ03 INMXZ03R	TRANSMIT encryption
INMXZ21R	TRANSMIT log data set pre-allocation
IRXINITX	REXX pre-environment initialization
IRXITTS IRXITMV	REXX post-environment initialization
IRXTERMX	REXX environment termination
Names service exit	Track changes made to Information Center Facility names directories
Session Manager	Initialization/stream monitoring/termination

Chapter 54. VTAM® Exits

VTAM Customization describes the exits listed in Table 18.

Table 18. VTAM Exits

Exit	Description
ISTAUCAG	Calculates and records time during which a terminal user or an application program is logged onto an application program
ISTAUCAT	Validates a logon request to an application program
ISTEXCVR	Provides ACF/VTAM with an ordered list of virtual routes for path selection to transmit data through network
ISTINCDT	User supplies supplementary tables to session-link unformatted system services table which ACF/VTAM uses to handle command input and message output
ISTINCLM	User can change IBM-supplied logon mode table which contains parameters representing protocols for telecommunications session or user can supply supplementary tables
ISTINCNO	User supplies supplementary tables to operation-level unformatted system services tables which handle commands from an ACF/VTAM operator and messages to an ACF/VTAM operator
ISTMGC00	Communication network management (CNM) table routes unsolicited network service requests to application programs to record and report maintenance statistics
ISTPUCWC	Virtual route pacing window size calculator specifies limits to data flow through a network to avoid congestion in nodes along a virtual route
ISTRACON	Module containing constants used to control functions not suitable for modification by operator command or start option

Part 4. MVS Converter / Interpreter Text Processing

Topics for This Section Appear as Follows:

The processing of a job control language (JCL) statement includes the stage of converting the JCL statement to C/I text, a form of data that the job entry subsystem (JES) and the job scheduler function of MVS both recognize. The converter takes the job's JCL, merges it with JCL from a procedure library, and converts the composite JCL into C/I text. The converter scans each JCL statement for syntax errors and issues appropriate error messages. The converter also resolves symbolic parameters and assigns default values. The converter / interpreter (C/I) text is further interpreted to build the necessary control blocks needed before the job can be scheduled for execution. See the JES2 exit, Exit 6 (converter/interpreter text scan) in *z/OS JES2 Installation Exits*, and the JES3 exit, IATUX03 (examine/modify converter/interpreter text created for JCL) in *z/OS JES3 Customization* for more information.

Chapter 55. Issuing Messages through JES Installation Exits

After all JCL statements are converted to C/I text, the converter calls a JES installation exit that allows the installation to issue messages to the JCLMSG data set and indicate whether or not the converter should fail the job. The installation must place any desired message in a converter message buffer (CNMB), which is mapped by the IEFCNMB mapping macro. See *z/OS MVS Data Areas* in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for more information on the CNMB data area.

The maximum length of a message for a single CNMB is 110 characters. If the CNMBMLEN field of a CNMB contains a value greater than 110, the message is truncated at 110 characters. You can chain multiple CNMBs together to pass longer messages. The CNMBNPTR field of the CNMB contains either the address of the next CNMB in the chain or zero if no other CNMBs follow.

Initializing the CNMB:

To initialize the converter message buffer (CNMB):

1. Obtain a buffer from a private area subpool, such as 230, to contain the CNMB. The storage must be obtained in key 1.

   You do not need to issue a FREEMAIN in the installation exit since the converter releases the storage for the CNMB after writing the messages to the JCLMSG data set. IBM recommends that you obtain buffer storage

   - in multiples of the CNMB mapping size (CNMBSIZE) when NOT issuing messages to the message data set, OR
   - in multiples of the CNMB mapping size plus the maximum size of a message text (CNMBSIZE + CNMBMAXL) when issuing messages to the message data set.

   This is to ensure that the converter releases the correct amount of storage and that no storage fragmentation occurs.
2. Set the following fields in all cases:
   - Set the CNMBID field to CNMBCID (C'CNMB').
   - Set the CNMBVER field to CNMBCVER.
   - Set the CNMBSUBP field with the hexadecimal equivalent of the subpool number from which the storage was obtained via GETMAIN.
3. Set the following fields if you are issuing a message to the message data set:
   - Set the CNMBMSG field with the message text.
   - Set the CNMBNPTR field to zeroes or to the address of the next CNMB if you are issuing a message that is longer than 110 characters.
   - Set the CNMBMLEN field with the length of the message text. The message text cannot be longer than 110 characters.
   - Set the CNMBLEN field based on how much storage was obtained:
     - If the storage is the size of the CNMB mapping plus the size of the message text, set the CNMBLEN field to CNMBSIZE + CNMBMLEN.
     - If the storage is the size of the CNMB mapping plus the **maximum** size of the message text, set the CNMBLEN field to CNMBSIZE + CNMBMAXL.
4. Set the following fields if you are failing the job:

- Set the CNMBFJOB bit on in the CNMBOPTS byte to indicate that the converter should fail the job. Setting the CNMBFJOB bit causes the converter to mark the job as failed when it regains control from the JES installation exit, and the job will not be run.
- Set the CNMBMLEN field to zero if the installation exit is not going to issue a message.

For more information on the JES installation text, see:

- Exit 6 in *z/OS JES2 Installation Exits*.
- IATUX03 in *z/OS JES3 Initialization and Tuning Guide*.

Chapter 56. Converter / Interpreter (C/I) Text Strings

Each C/I text record represents one JCL statement. This record, or text string, is built in an 8192-byte buffer. The length of the used data area of this buffer is specified by a 2-byte field at the beginning of the text string. **If modifications are made, it is the user's responsibility to ensure that the data in this length field is correct**.

The C/I text string for a record contains a hexadecimal value to indicate the statement type, such as JOB statement, EXEC statement, or last DD statement, among others. The text string also contains a specific key value (called the verb key) to identify the type of JCL statement (such as JOB, EXEC, DD).

The information for one type of JCL statement differs from the information for another, depending on the presence of parameters and subparameters. In general, however, the text strings adhere to a common format. Each C/I text record contains prefix information and an end-of-text indicator. In addition, key entries are defined for parameters in the text string. Positional information specified may be represented in the prefix and/or in the key entries.

Prefix Information

The prefix information contains the two-byte field at the beginning that specifies the length of the used data area of this record. The third byte is the statement type. Other prefix information relates specifically to the type of JCL statement (such as, "account number required" on a JOB statement or "SYSOUT data set" on a DD statement). The text string for a JOB statement contains seven bytes of prefix information; all other JCL statements have a prefix section containing five bytes.

Keyword Information

The keyword information defines the associated parameters on a JCL statement. The length of this section of the text string is variable depending on the number of parameters and subparameters. Each keyword (such as DSNAME or UNIT) has a one-byte hexadecimal key associated with it. These key values can be referenced with the mapping macro IEFVKEYS (data area ITK). For **each** of these keywords in the JCL statement, the following information (in this "key entry" format) is included in the text string:

Number of Parameters
: 1-byte hex key specifying the number of parameters.

Length
: 1-byte hex number specifying the length of the parameter. The high order bit is always off.

Parameter
: Variable length string containing the value of the specified parameter.

Number of Subparameters
: 1-byte hex number specifying the number of subparameters. The high order bit is always on.

Length
1-byte hex number specifying the length of the subparameter. The high order bit is always off.

Subparameter
Variable length string containing the value of the specified subparameter.

This key entry format is repeated for each keyword in the JCL statement.

End-of-text Information

The final entry in the text string is the end-of-text key. This entry is required by the MVS interpreter to indicate the end of the text string.

Chapter 57. Converter / Interpreter Text String Formats

The following section describe the formats of converter and interpreter text strings.

Prefix Format

The first three bytes of a text string always contain the string length and the statement type. The string length is a 2-byte length of the entire text string. The statement type is a 1-byte indicator of the type of text string.

The statement type is one of the following values:

X'01' JOB statement text string

X'02' EXEC statement text string

X'04' DD statement text string

X'08' PROC statement text string

X'10' Last statement for this step

X'20' JDT-defined verb string

X'40' JDT-defined JCL appears on this statement

X'80' Extended JCL statement types.

JOB String Prefix

String Length	Statement Type	Job Indicators	BLP Default	Job Verb Key

String Length
Two-byte length of the entire text string

Statement Type
One-byte indicator of the type of text string

Job Indicators
Two bytes of job-related information indicators

BLP Default
One byte containing the bypass label default

Job Verb Key
Verb key for the job text string (X'B4')

- The statement type value for a JOB statement is X'01'.
- Values for the two bytes of job indicators are as follows:
  - Byte 1

    X'01' Account number required

    X'02' Programmer name required

    X'04' Job has been failed

    X'08' Job has a SYSCHK DD statement

    X'10' Flush to restart step name

    X'20' Message header has been written

X'40' Region value is a default

X'80' JDT-defined JCL appears in this job's JCL

– Byte 2

X'01' JDT-defined JCL error in this job's JCL

X'02' Job is enabled to run with SWA located in virtual storage above 16 megabytes.

- The BLP default is one byte containing the bypass label default.

  X'01' No label

  X'10' Bypass label processing

- The job verb key for the JOB text string is always X'B4'.

EXEC String Prefix

String Length	Statement Type	EXEC Indicators	EXEC Verb Key

String Length
Two-byte length of the entire text string

Statement Type
One-byte indicator of the type of text string

EXEC indicators
One byte of EXEC-related information indicators

EXEC Verb Key
Verb key for the EXEC text string (X'94')

- The statement type value for an EXEC statement is X'02'.
- The EXEC indicator field contains one byte of EXEC-related information.

  X'01' Checkpoint restart EXEC statement

  X'02' Step has a STEPCAT DD

  X'04' Step has a STEPLIB DD

  X'08' Statement is from a procedure

  X'10' Step has no DD statements

  X'20' Statement invokes a procedure

- The verb key for the EXEC text string is always X'94'.

DD String Prefix

String Length	Statement Type	DD Indicators	DD Verb Key

String Length
Two-byte length of the entire text string

Statement Type
One-byte indicator of the type of text string

DD indicators
One byte of DD-related information indicators

DD Verb Key
Verb key for the DD text string (X'6E')

- The DD indicator field contains one byte of DD-related information.

  X'01' DUMMY specified

  X'02' DDNAME= specified

  X'04' DSNAME specified in quotation marks

  X'08' DYNAM specified

  X'10' SYSIN (DD * or DD DATA) data set

  X'20' SYSOUT data set

  X'40' SUBSYS= specified

  X'80' Statement is from a procedure
- The verb key for the DD text string is always X'6E'.

JDT String Prefix

String Length	Statement Type	JDT Indicators	JDT Verb Key

String Length
Two-byte length of the entire text string

Statement Type
One-byte indicator of the type of text string

JDT indicators
One byte of JDT-related information indicators (all reserved)

JDT Verb Key
Verb key for the JDT text string (X'BE')

- The JDT indicator field contains one byte of JDT-related information and is reserved.

  X'20' The statement is regenerated.

  X'40' Statement is generated.

  X'80' Statement is from a PROC.
- The verb key for the JDT-defined text string is always X'BE'.

Extended Statement Type String Prefix

String Length	Statement Type	Statement Indicators	Verb Key

String Length
Two-byte length of the entire text string

Statement Type
One-byte indicator of whether this is an extended statement type (always set to X'80')

Statement indicator
Two bytes of extended statement type indicators

Verb Key
Verb key for the extended statement text string

- The statement indicator field contains one byte that indicates one of the following statement type values:

X'80' IF statement

X'40' ELSE statement

X'20' ENDIF statement

- The verb key for the extended statement type text string is equal to the extended statement type value.

Positional Format

Positional parameters specified may be represented in the prefix and/or in the key entries depending on the particular parameter. The following section describes the manner in which each positional parameter is represented.

JOB String Positional Parameters

```
//JOB1   JOB  (1234,ABCD),'John Doe'
```

Key	No. Parms	Length Jobname	Jobname	Sublist/ no. elements	Length Account Par 1	Account Info. Par 1	Length Account Par 2	Account Info. Par 2	Length Name	Programmer Name
B4	03	04	JOB1	82	04	1234	04	ABCD	08	John Doe

- The statement label is the first parameter on the job verb key.
- Accounting information is the second parameter on the job verb key.
- Programmer name is the third parameter on the job verb key.

```
//JOB1   JOB  (1234),'John Doe'
```

Key	No. Parms	Length Jobname	Jobname	Sublist / no. elements	Length Account parm#1	Account Info.	Length Name	Programmer Name
B4	03	04	JOB1	81	04	1234	08	John Doe

- The statement label is the first parameter on the JOB verb key.
- Accounting information is the second parameter on the JOB verb key.

  Note: Enclosing the accounting information within parentheses indicates that the information is in the form of a subparameter list. In this case, the subparameter list has one element (1234).
- Programmer name is the third parameter on the JOB verb key.

EXEC String Positional Parameters

The verb key for an EXEC statement is always X'94'. When executing a procedure, the positional parameters associated with the EXEC statement depend on whether the statement has been coded with the PROC= keyword or not.

- When no PROC= keyword is used, the text string for the statement contains two positional parameters on the EXEC verb key.

  ```
  //STEP1  EXEC PROC1
  ```

  1. The statement label (STEP1) is the first parameter on the EXEC verb key.
  2. The PROC name (PROC1) is the second parameter on the EXEC verb key.

Verb Key	Number Parameters	Label Length	Label	Proc Length	Proc Name
94	02	05	STEP1	05	PROC1

- When the PROC= keyword is used, the text string contains a verb key for the EXEC statement (X'94') and also a PROC key for the PROC= keyword (X'8B'). The text string for the statement contains one positional parameter on each key.

```
//STEP1  EXEC PROC=PROC1
```

  1. The statement label is the first parameter on the EXEC verb key (X' 94').
  2. The PROC name is the first parameter on the PROC key (X'8B').

Verb Key	Number Parameters	Label Length	Label	Proc Key	Number Parameters	Proc Length	Proc Name
94	01	05	STEP1	8B	01	05	PROC1

DD String Positional Parameters

There are two instances when a positional parameter is represented by key entries in the text string for a DD statement. When a DD statement specifies DD DYNAM, the DD indicator in the prefix is set to X'08'. When it specifies DD DUMMY, the DD indicator is set to X'01'. In both of these cases, a key entry containing the respective key value and one parameter of length zero is added to the text string.

If a DD statement specifies DD * or DD DATA, the DD indicator in the prefix is set to X'10'. No other key entry is defined in the text string for these positional parameters.

Text Format for JDT-defined JCL

- JDT-defined verbs

```
//OUTPUT1 OUTPUT  .....
```

Verb Key	Number Parameters	Verb Length	Verb	Label Length	Label
BE	02	06	OUTPUT	07	OUTPUT1

-
  - The JDT-defined verb is the first parameter on the verb key (X'BE').
  - The statement label is the second parameter on the verb key (X'BE').
- JDT-defined keywords

```
//DD1     DD  ACCODE=A
```

Long Parameters: The length of some parameters, such as a pathname for an hierarchical file, can be up to 255 bytes. Any parameter with a maximum length that is over 127 bytes is represented in the key entry format as a subparameter list within the key for the parameter. Each subparameter has a maximum length of 127 bytes.

The PATH parameter is used to specify the pathname:

```
//ddname DD PATH=pathname
```

Example 1: In the following example, the PATH parameter has 6 characters:

```
//DD2  DD PATH='/name1'
```

Even though the pathname parameter is only 6 bytes, it must be represented as a subparameter list because the maximum length of the pathname can be 255 bytes.

Table 19. Text Format for Long Parameters: Example 1

Key	Number Parameters	Key Length	JDT-Defined Key for Specified Keyword	Number Sub-parameters	Parameter Length	Parameter Value
1A	02	02	X'8017'	81	6	X'61D5C1D4C5F1'

Example 2: In this example, the PATH parameter has 130 characters:

```
//DD2 DD PATH='/NAME11111/NAME22222/NAME33333/NAME44444/NAME55555/NAME6
//             6666/NAME77777/NAME88888/NAME99999/NAME00000/NAME11111/N
//             AME22222/NAME33333'
```

It would be represented as shown in Table 20.

Table 20. Text Format for Long Parameters: Example 2

Key	Number Parameters	Key Length	JDT-Defined Key for Specified Keyword	Number Sub-parameters	Parameter Length	Parameter Value
1A	02	02	X'8017'	82	7F	X'61D5C1D4C5F1F1F1F1F1,,,61D5C1D4C5F3F3'
					03	X'F3F3F3'

Because only 127 characters can fit in a single subparameter field, the parameter must be represented as two subparameters. Therefore, the `Num of Subparameters` field is 82.

- The first subparameter is X'7F' (127 decimal) bytes long, and its value in the C/I text is the hex string representing the first 127 characters of the PATH parameter value in the JCL.
- The second subparameter is 3 bytes long, and its value in the text is the hex string representing the last 3 bytes of the PATH parameter value.

Pre-MVS/ESA SP 4.1: For JCL converted on a pre-MVS/ESA SP 4.1 system, the JDT-defined keyword is the first parameter on the JDT key (X'1A').

Key	Number of Parameters	Keyword Length	Keyword	Parameter Length	Parameter
1A	02	06	ACCODE	01	A

- The parameter on the JDT-defined keyword is the second parameter on the JDT key (X'1A').

MVS/ESA SP 4.1: For JCL converted on an MVS/ESA SP 4.1 system, the JDT-defined key for the specified JDT keyword is the first parameter on the JDT key (X'1A'). For JDT-defined keywords on the DD statement, refer to the macro IEFSJKEY. For JDT-defined keywords on the OUTPUT statement, refer to the macro IEFDOKEY.

Key	Number of Parameters	Key Length	JDT-Defined Key for the Specified Keyword	Parameter Length	Parameter
1A	02	02	X'8001'	01	A

- The parameter on the JDT-defined keyword is the second parameter on the JDT key (X'1A').

Extended Statement Type String Positional Parameters

```
//IFBAD  IF  (STEP1.RC > 4)  THEN
```

Key	BF
Number of Name Parms	00 02
Number of Name Qualifiers	01
Length of Qualifier	05
Subject of Test	03
Length Compared Value	04
Compared Value	4
Type of Comparator	0C
Number of Comparators	01
Length	01
Comparator	>

- The qualifier specifies the step and procedure level of the statement.
- The possible values for the "Subject of Test" field are:

  X'01' Job level return code

  X'02' Step level return code (within the procedure)

  X'03' Return code

  X'04' Job abend code

  X'05' Step abend code (within the procedure)

  X'06' User abend code

  X'07' System abend code

  X'08' Job abend code

  X'09' Step abend code

  X'0A' Processed/run
- The "Type of Comparator" field can be set to either of the following values:

  X'0B' Boolean comparator (such as | or &);

  X'0C' Mathematical operator (such as < or >)

Key Entry Format Examples

```
SPACE=(TRK,(30,10))
```

Key	Number Parms	Length	Parameter	Number Subparm	Length	Parameter	Length	Parameter
47	02	03	TRK	82	02	F3F0	02	F1F0

The key X'47' represents the keyword SPACE=, which has two parameters — TRK and (30,10). The first parameter, TRK, has a length of 3. The second parameter is made up of two subparameters and the high order bit of that field is on (X'82'). The length of each subparameter is 2; the values are given in EBCDIC.

VOL=SER=111111

Key	Number Parms	Key	Number Parms	Length	Parameter
43	00	4F	01	06	F1F1F1F1F1F1

The key X'43' represents the keyword VOL=. The subparameter keyword, SER=, is considered a minor keyword and is assigned a unique key entry. Therefore SER= will not be defined in the key entry for the keyword (VOL=) on which it appears. VOL= (key X'43') then has no parameters. SER= (key X' 4F') has one parameter with a length of 6. The value of this parameter (111111) is given in EBCDIC.

Referral Type Data

Referral type data is represented in the key entry format as subparameter fields. The pieces of the parameter are defined as delimited by periods. Each piece of the referred-to data is identified by its length and name in the order in which it appears in the statement. The asterisk is the first subparameter defined.

```
//DD1     DD  DCB=*.STEP1.DD1
```

Key	Number Parameters	Number Subparms	Length	*	Length Piece 1	Piece 1	Length Piece 2	Piece 2
40	01	83	01	*	05	STEP1	03	DD1

- The asterisk is the first subparameter.
- The remaining subparameters are the pieces of the referred-to data delimited by periods.

Note: In the "number of subparameters" field, the high-order bit is always set on.

Data Set Name with Member Name

A data set name that includes a member is represented in the key entry format as subparameter fields within the key for the parameter. The data set is identified as the first subparameter; the member name is the second subparameter.

```
//DD1     DD  DSN=THIS.DATA1(MEMBER1)
```

Key	Number Parms	Number Subparms	Length DS Name	Data set Name	Length Member	Member Name
4A	01	82	0A	THIS.DATA1	07	MEMBER1

- The data set name is the first subparameter.
- The member name is the second subparameter.

Overrides of Parameters

Keywords that override EXEC statement keywords within a procedure appear in the text string for the EXEC statement for the procedure. The values specified for the overridden keywords are applied to the EXEC statement during conversion.

```
//                EXEC PROC=PROCA,TIME.STEP1=2,PARM.STEP2='ABCD'
```

Proc Key	Number Parameters	Proc Length	Proc Name
8B	01	05	PROCA

Key	Number Parameters	Step Length	Step Name	Parm Length	Parm
8F	02	05	STEP1	01	F2

Key	Number Parameters	Step Length	Step Name	Parm Length	Parm
8E	02	05	STEP2	04	ABCD

DD statements that override DD statements within a procedure are merged with the overridden statement as the statements are converted. The resulting text string contains the result of the merged statements. The merging of the statements keeps all the keywords present on the overridden statement as long as they are not explicitly overridden or are not mutually exclusive with a keyword already specified on the overriding statement.

```
//DD1 DD SYSOUT=A,DCB=(LRECL=133,DSORG=PS)
```

is overridden by

```
//STEP1.DD1 DD DSN=XYZ,DCB=LRECL=80,DISP=SHR
```

The keywords are merged so that overridden parameters are replaced and mutually exclusive parameters are nullified. The resulting text string contains

```
//DD1 DD DSN=XYZ,DCB=(LRECL=80,DSORG=PS),DISP=SHR
```

End-of-text Format

This final entry, after return from the installation's converter / interpreter (C/I) text exit, if any, (that is, Exit 6 in JES2, Exit IATUX03 in JES3), signifies the end of the text string with an end-of-text key. (Prior to and within the C/I text exit, if any, the end-of-text format is just the **FE** key, with nothing following it.)

Key	Number Parms	Length	Proc Level	Length	Statement Number
FE	02	01		04	

The procedure nesting level is a value between 0 and 15 inclusive, and is obtained as follows:

- JOB-level statements (not in a procedure) are level 0.
- Statements within the first procedure level are level 1.
- The nesting level is incremental for each successive level of procedure nesting.
- The nesting level is not affected by INCLUDE statement nesting.

See *z/OS MVS JCL Reference* for more information on procedure nesting.

Note that the system ignores JCL comment statements when computing the statement number.

Examples of MVS/CI Text Strings

```
//SYSUT1 DD DSNAME=LINKEDIT.WORK,UNIT=3380,SPACE=(TRK,(30,10)),
//          VOL=SER=111111
```

Prefix and Statement Label (SYSUT1)

String Length	Type	DD Ind.	DD Verb Key	Number Parms	Length	Parameter	.....
003D	04	00	6E	01	06	SYSUT1	

Examples of MVS/CI Text Strings

DSNAME=LINKEDIT.WORK,

Key	Number Parms	Length	Parameter	.....
4A	01	0D	LINKEDIT.WORK	

UNIT=3380,

Key	Number Parms	Length	Parameter	.....
41	01	04	F3F3F8F0	

SPACE=(TRK,(30,10)),

Key	Number Parms	Length	Parameter	Number Subparm	Length	Parameter	Length	Parameter	.....
47	02	03	TRK	82	02	F3F0	02	F1F0	

VOL=SER=111111

Key	Number Parms	Key	Number	Length	Parameter	.....
43	00	4F	01	06	F1F1F1F1F1F1	

User References

You can find additional information about C/I text in MVS macros. The C/I text format is mapped in the macro IEFTXTFT. The table of converter/interpreter key definitions is mapped in the macro IEFVKEYS (data area ITK).

Chapter 58. Modifying Converter / Interpreter Text

Both JES2 and JES3 provide installation exit points for scanning the text created by the converter. You can use these exits to decide whether JES should cancel the job or allow it to continue normally. Your routine also can modify the C/I text. You cannot add text for an additional JCL statement, but you can add parameters to an existing JCL statement. You cannot delete the text for an entire JCL statement, but you can delete some of the parameters on that statement.

Use Caution When Modifying C/I Text. At the exit point at which the text is made available to your routine, the data already has been validated for syntax and for the proper keywords. The converter **does not** repeat this validation process after any modifications that you make. Therefore, there is no way to correct any error that a modification to the C/I text causes, and the job might fail at a later point in processing.

MVS creates the C/I text and uses it in its preparation for executing your job. IBM recommends that you do not modify this text. If you decide to do so, use the following guidelines:

The interpreter needs to have all the correct lengths, the number of parameters, and the end-of-text key (X'FE') in order to parse the text string. You must ensure that the length of the entire text string (in the prefix) and the values for the length and number of parameters (in the key entries) are updated to reflect any modifications that you make.

More information about how you can use installation exits to modify C/I text is available in the following documents :

- *z/OS JES3 Customization*
- *z/OS JES2 Installation Exits*.

Part 5. Testing SMF Exit Routines

Topics for This Section Appear as Follows:

This section describes one method of testing user-written SMF exit routines, the TESTEXIT procedure in SYS1.SAMPLIB. This procedure contains an assembler language source program (also named TESTEXIT) which attaches the data generator utility program (IEBDG) to create sample parameter lists for all user-written exit routines except IEFU29. (The TESTEXIT procedure creates the parameter list for the IEFU29 exit routine without using the data generator utility program.) The source program then calls each user-written exit routine being tested, and passes the appropriate parameter list to it.

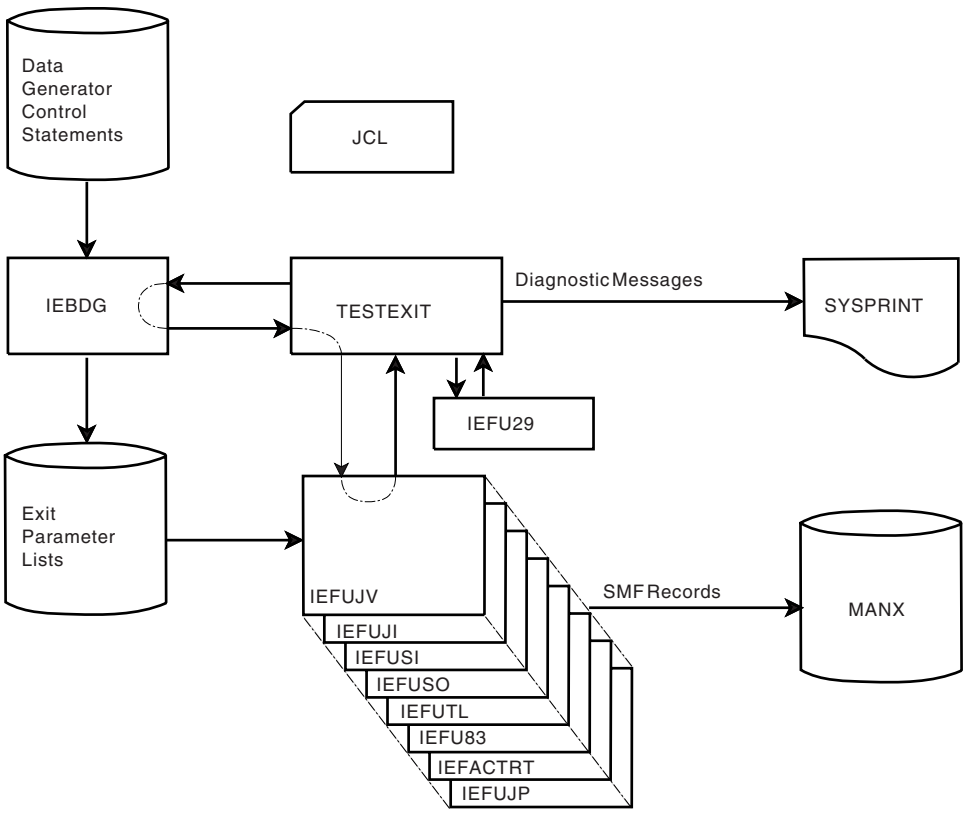

Figure 27. TESTEXIT Input/Output and Control Flow

Before using the TESTEXIT procedure:

1. Fulfill the following user-written exit routine testing requirements:
   - Specify a user subpool (0-127) in all GETMAIN macro instructions included in the routines.

- Provide a special SMFWTM macro instruction in all routines that use the macro.
- Place the routines in a partitioned data set.

2. Obtain the TESTEXIT procedure from SYS1.SAMPLIB.
3. Modify the procedure to meet the installation's testing requirements.

Chapter 59. TESTEXIT Exit Routine Requirements

Before using the TESTEXIT procedure, fulfill the following exit routine requirements:

- Specify a user subpool (0-127) in all GETMAIN macro instructions included in your routines.
- Provide a special SMFWTM macro instruction in all exit routines that use the macro. The special macro definition writes to the TESTEXIT data set defined by the DD statement named MANX. (With the normal SMFWTM macro instruction the data is written to the active SMF data set.) Using this macro definition, then, data is processed without accessing the system data on the active SMF data set. When testing is completed, remove the macro definition.

  Figure 28 shows the SMFWTM macro instruction that is required for using the TESTEXIT procedure.

```
        MACRO
&NAME   SMFWTM  &MSGAD
        AIF     ('&MSGAD'EQ');E1
        AIF     ('&MSGAD'EQ'(1)';BAL
        AIF     ('&MSGAD'(1,1)EQ(');REGA
        AGO     .LODIT
.E1     MNOTE   '***NO OPERAND SPECIFIED***'
        MEXIT
.BAL    ANOP
        CNOP    0.4
&NAME   BAL     15,*+8
.LIST   DC      V(TSMFWTM)
        L       15,0(15)
        BALR    14,15
        MEXIT
.REGA   ANOP
&NAME   LR      1,&MSGAD(1)
        CNOP    0,4
        BAL     15,*+8
        AGO     .LIST
.LODIT  ANOP
&NAME   LA      1,&MSGAD
        CNOP    0,4
        BAL     15,*+8
        AGO     .LIST
        MEND
```

Figure 28. SMFWTM Macro Definition Required for Using TESTEXIT

- Place the exit routines in a partitioned data set named EXITLIB. Figure 29 on page 378 shows sample JCL for entering the routines into EXITLIB.

```
//UPDTE     JOB    MSGLEVEL=1
//          EXEC   PGM=IEBUPDTE,PARM=NEW
//SYSUT2    DD     DSNAME=EXITLIB,VOLUME=SER=338000,
//          UNIT=3380,SPACE=(TRK,(10,3,1)),DISP=(,KEEP),
//          DCB=(LRECL=80,BLKSIZE=400,RECFM=FB)
//SYSPRINT  DD     SYSOUT=A
//SYSIN     DD     DATA
./ ADD      NAME=IEFUJV

(IEFUJV object deck)
./ ADD      NAME=IEFUJI

(IEFUJI object deck)
./ ADD      NAME=IEFUSI

(IEFUSI object deck)
./ ADD      NAME=IEFUTL

(IEFUTL object deck)
./ ADD      NAME=IEFUSO

(IEDUSO object deck)
./ ADD      NAME=IEFU83

(IEFU83 object deck)
./ ADD      NAME=IEFACTRT

(IEFACTRT object deck)
./ ADD     NAME=IEFUJP

(IEFUJP object deck)
./ ADD    NAME=IEFU29

(IEFU29 object deck)
./ ADD     NAME=IEFU84

(IEFU84 object deck)
./ ENDUP
/*
```

Figure 29. Sample JCL for Entering User-Written Exit Routines into EXITLIB

Obtaining TESTEXIT from SYS1.SAMPLIB

Figure 30 shows sample JCL for obtaining a punched deck of TESTEXIT from SYS1.SAMPLIB.

```
//PUNCH     JOB       MSGLEVEL=1
//          EXEC      PGM=IEBPTPCH
//SYSPRINT  DD        SYSOUT=A
//SYSUT1    DD        DSNAME=SYS1.SAMPLIB,DISP=(OLD,KEEP),
//          UNIT=xxxx,VOLUME=SER=xxxxxx[1]
//SYSUT2    DD        UNIT=2540-2
//SYSIN     DD        *
            PUNCH     TYPORG=PO,MAXNAME=1,MAXFLDS=1
            MEMBER    NAME=TESTEXIT
            RECORD    FIELD=(80)
```

[1]The volume and unit parameters depend on your installation's request.

Figure 30. Sample JCL for Obtaining a Punched Deck of TESTEXIT

Modifying the TESTEXIT Procedure

"Sample JCL for Executing TESTEXIT" shows sample JCL for executing the TESTEXIT procedure.

Sample JCL for Executing TESTEXIT

```
//TESTEXIT       JOB  MSGLEVEL=1
//TEST           EXEC ASMFCL
//ASM.SYSIN      DD     *
(TESTEXIT Source Module)
/*
//LKED.SYSLMOD   DD   DSNAME=TESTLIB,VOLUME=SER=338000.
//               UNIT=3380,SPACE=(TRK,(5,2,1))),
//               DISP=(NEW,KEEP)
//LKED.EXITS     DD   DSNAME=EXITLIB,VOLUME=SER=338000.
//               UNIT=3380,DISP=OLD
//LKED.SYSIN     DD    *
  INCLUDE EXITS(IEFUJV,IEFUJI,IEFUSI,IEFUTL,IEFUSO,
                IEFU83,IEFACTRT,IEFUJP,IEFU29,IEFU84)
  ENTRY TESTEXIT
  NAME TESTEXIT
/*
//DATAGEN            JOB  MSGLEVEL=1
//                   EXEC PGM=IEBGENER
//SYSUT2             DD   DSNAME=DGINPUT,UNIT=3380,DISP=(,KEEP),
//                   VOLUME=SER=338000,SPACE=(TRK,(10,5,1)),
//                   DCB=(LRECL=80,BLKSIZE=400,RECFM=FB)
//SYSPRING           DD   SYSOUT=A
//SYSIN              DD   *,DLM=XX
   GENERATE MAXNAME=9,MAXGPS=0
   MEMBER   NAME=UJV
   RECORD   IDENT=(6,'ENDUJV',1)
   MEMBER   NAME=UJI
   RECORD   IDENT=(6,'ENDUJI',1)
   MEMBER   NAME=USI
   RECORD   IDENT=(6,'ENDUSI',1)
   MEMBER   NAME=UTL
   RECORD   IDENT=(6,'ENDUTL',1)
   MEMBER   NAME=U83
   RECORD   IDENT=(6,'ENDU83',1)
   MEMBER   NAME=ACT
   RECORD   IDENT=(6,'ENDACT',1)
   MEMBER   NAME=USO
   RECORD   IDENT=(6,'ENDUSO',1)
   MEMBER   NAME=UJP
   RECORD   IDENT=(6,'ENDUJP',1)
   MEMBER   NAME=U84
   RECORD   IDENT=(6,'ENDU84',1)
XX
//SYSUT1             DD   DATA,DLM=YY
                     DSD  OUTPUT=(OUTUJV)
                          (IEBDG Control Statements for IEFUJV)
ENDUJV               END
                     DSD  OUTPUT=(OUTUJI)
                     (IEBDG Control Statements for IEFUJI)
ENDUJI               END
                     DSD  OUTPUT=(OUTUSI)
                     (IEBDG Control Statements for IEFUSI)
ENDUSI             END
                   DSD    OUTPUT=(OUTUTL)
                   (IEBDG Control Statements for IEFUTL)
ENDUTL             END
                   DSD    OUTPUT=(OUTU83)
                   (IEBDG Control Statements for IEFU83)
ENDU83             END
                   DSD    OUTPUT=(OUTACT)
```

```
                (IEBDG Control Statements for IEFACTRT)
ENDACT          END
                DSD    OUTPUT=(OUTUSO)
                (IEBDG Control Statements for IEFUSO)
ENDUSO          END
                DSD    OUTPUT=(OUTUJP)
                (IEBDG Control Statements for IEFUJP)
ENDUJP          END
                DSD    OUTPUT=(OUTU84)
                (IEBDG Control Statements for IEFU84)
ENDU84          END
YY
//TESTING       JOB    MSGLEVEL=1
//JOBLIB        DD     DSNAME=TESTLIB,VOLUME=SER=338000,
//              UNIT=3380,DISP=(OLD,KEEP)
//              EXEC   PGM=TESTEXIT,
// PARM='UJV=25,UJI=8,USI=8,USO=5,UTL=5,U83=12,ACT=2,UJP=2,U29=2,U84=12'
//INUJV         DD     DSNAME=DGINPUT(UJV),DCB=(LRECL=80,
//              BLKSIZE=400,RECFM=FB),DISP=(OLD,PASS),
//              UNIT=3380,VOLUME=SER=338000
//INUJI         DD     DSNAME=DGINPUT(UJI),DCB=(LRECL=80,
//              BLKSIZE=400,RECFM=FB),DISP=(OLD,PASS),
//              UNIT=3380,VOLUME=SER=338000
//INUSI         DD     DSNAME=DGINPUT(USI),DCB=(LRECL=80,
//              BLKSIZE=400,RECFM=FB),DISP=(OLD,PASS),
//              UNIT=3380,VOLUME=SER=338000
//INUSO         DD     DSNAME=DGINPUT(USO),DCB=(RECL=80,
//              BLKSIZE=400,RECFM=FB),DISP=(OLD,PASS),
//              UNIT=3380,VOLUME=SER=338000
//INUTL         DD     DSNAME=DGINPUT(UTL),DCB=(LRECL=80,
//              BLKSIZE=400,RECFM=FB),DISP=(OLD,PASS),
//              UNIT=3380,VOLUME=SSER=338000
//INU83         DD     DSNAME=DGINPUT(U83),DCB=(RECL=80,
//              BLKSIZE=400,RECFM=FB),DISP=(OLD,PASS),
//              UNIT=3380,VOLUME=ser=338000
//INACT         DD     DSNAME=DGINPUT(ACT),DCB=(RECL=80,
//              BLKSIZE=400,RECFM=FB),DISP=(OLD,PASS),
//              UNIT=3380,VOLUME=SER=338000
//INUJP         DD     DSNAME=DGINPUT(UJP),DCB=(LRECL=80,
//              BLKSIZE=400,RECFM=FB),DISP=(OLD,PASS),
//              UNIT=3380,VOLUME=SER=338000
//INU84         DD DSNAME=DGINPUT(U84),DCB=(RECL=80,
//              BLKSIZE=400,RECFM=FB),DISP=(OLD,PASS),
//              UNIT=3380,VOLUME=SER=338000
//OUTUJV      DD    DSNAME=UJV(OUT),UNIT=3380,DISP=(,PASS),
//            SPACE=9TRK,(10,5,1)),VOLUME=SER=338000,
//            DCB=(LRECL=80,BLKSIZE=400,RECFM=FB)
//OUTUJI      DD    DSNAME=UJI(OUT),UNIT=3380,DISP=(,PASS),
//            SPACE=(TRK,(10,5,1)),VOLUME=SER=338000,
//            DCB=(LRECL=80,BLKSIZE=400,RECFM=FB)
//OUTUSI      DD    DSNAME=USI(OUT),UNIT=3380,DISP=(,PASS),
//            SPACE=(TRK,(10,5,1)),VOLUME=SER=338000,
//            DCB=(LRECL=80,BLKSIZE=400,RECFM=FB)
//OUTUSO      DD    DSNAME=USO(OUT),UNIT=3380,DISP=(,PASS),
//            SPACE=(TRK(10,5,1)),VOLUME=SER=338000,
//OUTUTL      DD    DSNAME=UTL(OUT),UNIT=3380,DISP=(,PASS),
//            SPACE=(TRK,(10,51,1)),VOLUME=SER=338000,
//            DCB=(LRECL=80,BLKSIZE=400,RECFM=FB)
//OUTU83      DD    DSNAME=U83(OUT),UNIT=3380,DISP=(,PASS),
//            SPACE=(TRK,(10,5,1)),VOLUME=SER=338000,
//            DCB=(LRECL=130,BLKSIZE=130,RECFM=FB)
//OUTACT      DD    DSNAME=ACT(OUT),UNIT=3380,DISP=(,PASS),
//            SPACE=(TRK,(10,5,1)),VOLUME=SER=338000,
//            DCB=(RECL=180,BLKSIZE=180,RECFM=FB)
//OUTUJP      DD    DSNAME=UJP(OUT),UNIT=3380,DISP=(,PASS),
//            SPACE=(TRK,(10,5,1)),VOLUME=SER=338000,
//            DCB=(LRECL=130,BLKSIZE=130,RECFM=FB)
```

```
//OUTU84      DD DSNAME=U84(OUT),UNIT=3380,DISP=(,PASS),
//            SPACE=(TRK,(10,5,1)),VOLUME=SER=338000,
//            DCB=(LRECL=130,BLKSIZE=130,RECFM=FB)
//MANX        DD    UNIT=3380,VOLUME=SER=338000,DSN=MANX,
//            SPACE=(TRK,(3,1)),DISP=(NEW,KEEP),
//            DCB=(BLKSIZE=200,LRECL=196)
//SYSPRINT    DD    SYSOUT=a,dcb=(BLKSIZE=136,LRECL=132)
//DGPRINT     DD    SYSOUT=A
//SYSABEND    DD    SYSOUT=A
/*
```

The following summarizes the operations performed by the procedure shown in "Sample JCL for Executing TESTEXIT" on page 379:

- The TESTEXIT job assembles the TESTEXIT source program (not illustrated in the figure) and linkedits it with the exit routines being tested. (Note that the exit routines must reside in EXITLIB, a partitioned data set.)
- The DATAGEN job, using the IEBGENER utility program, creates a partitioned data set (DGINPUT) containing control statements for the IEBDG utility program, which will be attached by the TESTEXIT source program.
- The TESTING job includes the execution of the TESTEXIT source program.

Use the TESTEXIT procedure provided in SYS1.SAMPLIB to linkedit the example exit routines in SYS1.SAMPLIB, generate sample parameter lists, and test the sample exit routines. To adapt the TESTEXIT procedure to your installation's testing requirements, however, note the following modifications:

- The TESTEXIT job shown in "Sample JCL for Executing TESTEXIT" on page 379 linkedits the TESTEXIT source program with the exit routines. The TESTEXIT procedure in SYS1.SAMPLIB contains ten exit names in the INCLUDE statement. However, when you use the TESTEXIT procedure your INCLUDE statement should contain only the names of the exit routines you are testing.
- The DATAGEN job shown in "Sample JCL for Executing TESTEXIT" on page 379 creates a partitioned data set containing the IEBDG control statements that generate samples of standard parameter lists. The TESTEXIT procedure contains the control statements for nine exits. Note that control statements are not required for the IEFU29 exit routine because the TESTEXIT procedure creates the parameters needed to test that routine. When using the TESTEXIT procedure you should include only those statements needed for the routine you are testing. When testing for special conditions or required additional test parameters, you must make appropriate modifications and additions to the control statements.

  You must supply the control statements in such an order that the records the IEBDG utility program generates later will be grouped as complete parameter lists that conform in length and format to the exit parameters defined earlier in this chapter . (Be sure to include the entry code passed to exits IEFUTL and IEFACTRT in register 0 as a one-byte parameter at the end of the parameter lists for those exits.) For detailed information on using IEBDG control statements, see *z/OS DFSMSdfp Utilities*.
- The TESTING job shown in "Sample JCL for Executing TESTEXIT" on page 379 includes the execution of the TESTEXIT source program. Values for the PARM parameter of the EXEC statement specify which exit routines are to be tested and the number of times each is to be tested. The TESTEXIT procedure in SYS1.SAMPLIB contains the parameters to test ten exits. However, when you use the TESTEXIT procedure you should include only the parameters for the routines you are testing. This parameter has the format:

```
PARM='xxx=nnn,...,xxx=nnn'
```

where:

xxx
is an exit routine identifier.

nnn
is the number of times an exit routine is to be tested (the maximum value is 255).

The DD statements to be included depend upon the exit routines being tested. The TESTEXIT procedure contains DD statements for nine exits as shown in the sample ("Sample JCL for Executing TESTEXIT" on page 379). When you use the TESTEXIT procedure you should include only the DD statements for exits you are testing. DD statements are not required for the IEFU29 exit. Table 21 shows the exit-routine identifiers, specified on the EXEC statement, and the DD statements that you must include for each exit routine being tested.

Table 21. Parameters and DD Statements for Executing TESTEXIT

Exit Routine	Identifier	DD Statements
IEFUJV	UJV	INUJV, OUTUJV
IEFUJI	UJI	INUJI, OUTUJI
IEFUSI	USI	INUSI, OUTUSI
IEFUTL	UTL	INUTL, OUTUTL
IEFUSO	USO	INUSO, OUTUSO
IEFU83	U83	INU83, OUTU83
IEFACTRT	ACT	INACT, OUTACT
IEFUJP	UJP	INUJP, OUTUJP
IEFU29	U29	Not required
IEFU84	U84	INU84, OUTU84
Any		MANX,SYSPRINT,DGPRING,SYSABEND

You must include (in the JCL for the TESTEXIT procedure) the DD statements for any other data sets the exit routines use.

Part 6. SMF Exit — System Interface Diagrams

This section contains diagrams that show the system interface(s) for the SMF exit routines listed below. Each diagram illustrates the general flow of events that occur before and after the exit routine receives control. Note that the diagrams do not indicate the specific control path between system modules.

The system interfaces for the following SMF exits are illustrated:

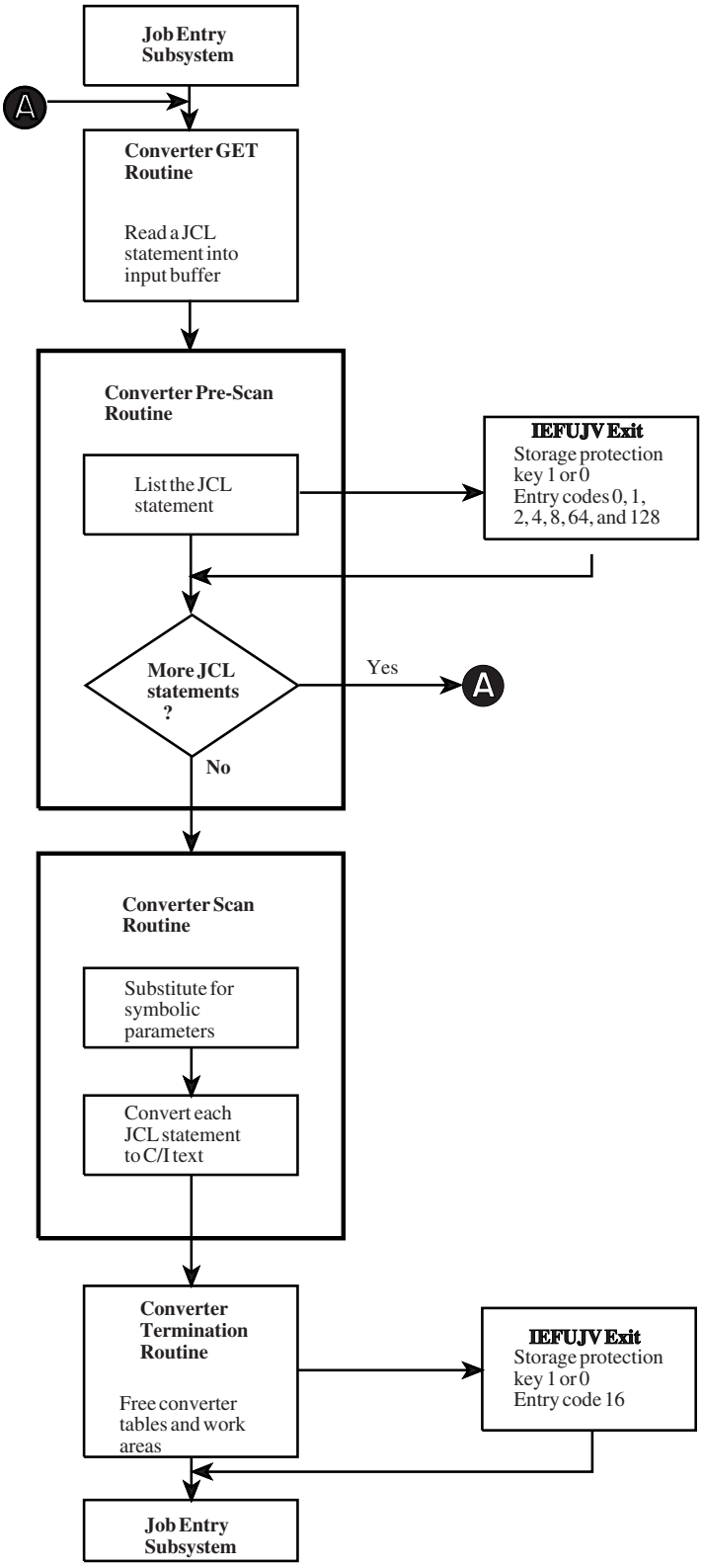

Figure 31. IEFUJV — Job Validation Exit (Converter)

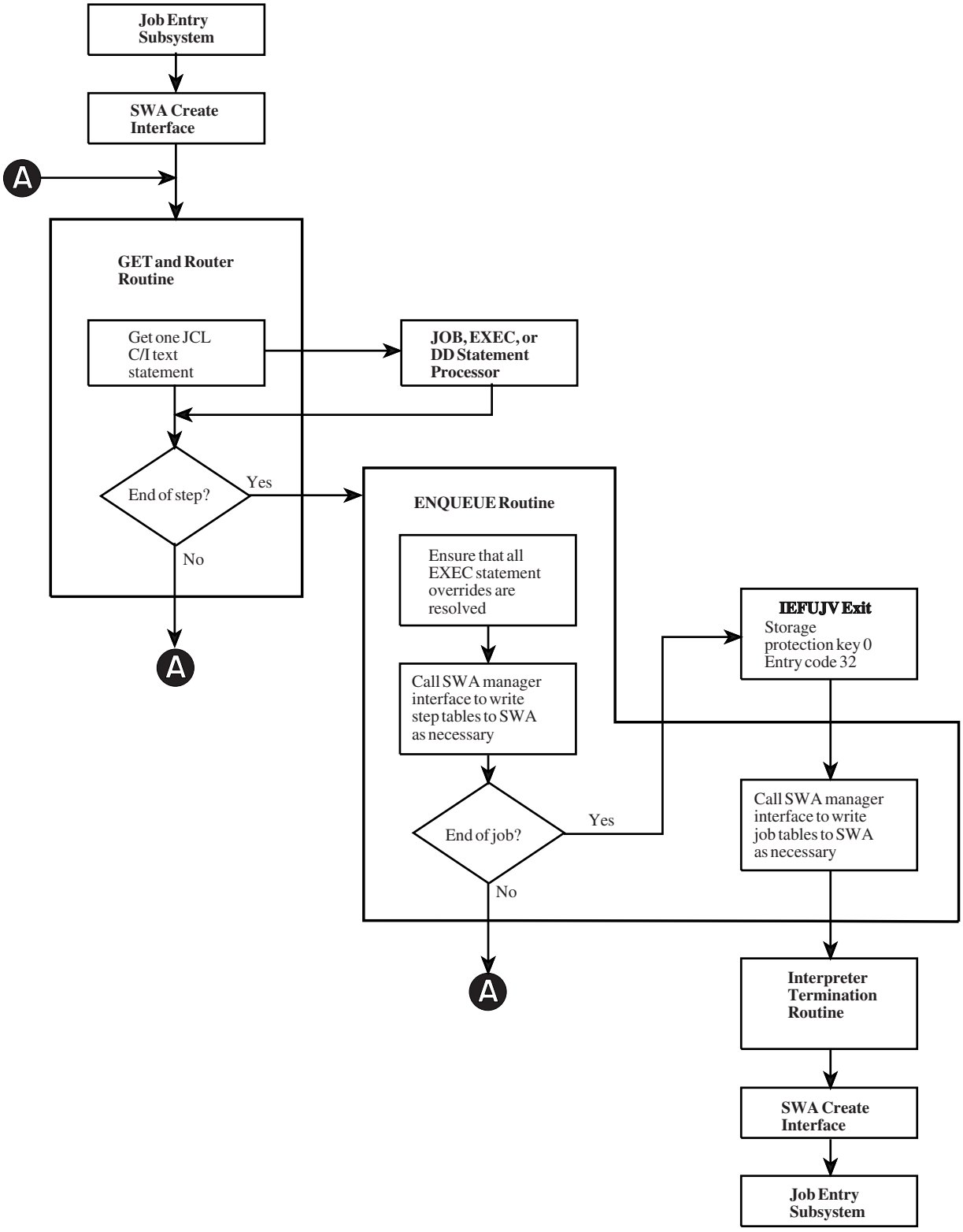

Figure 32. IEFUJV — Job Validation Exit (Interpreter)

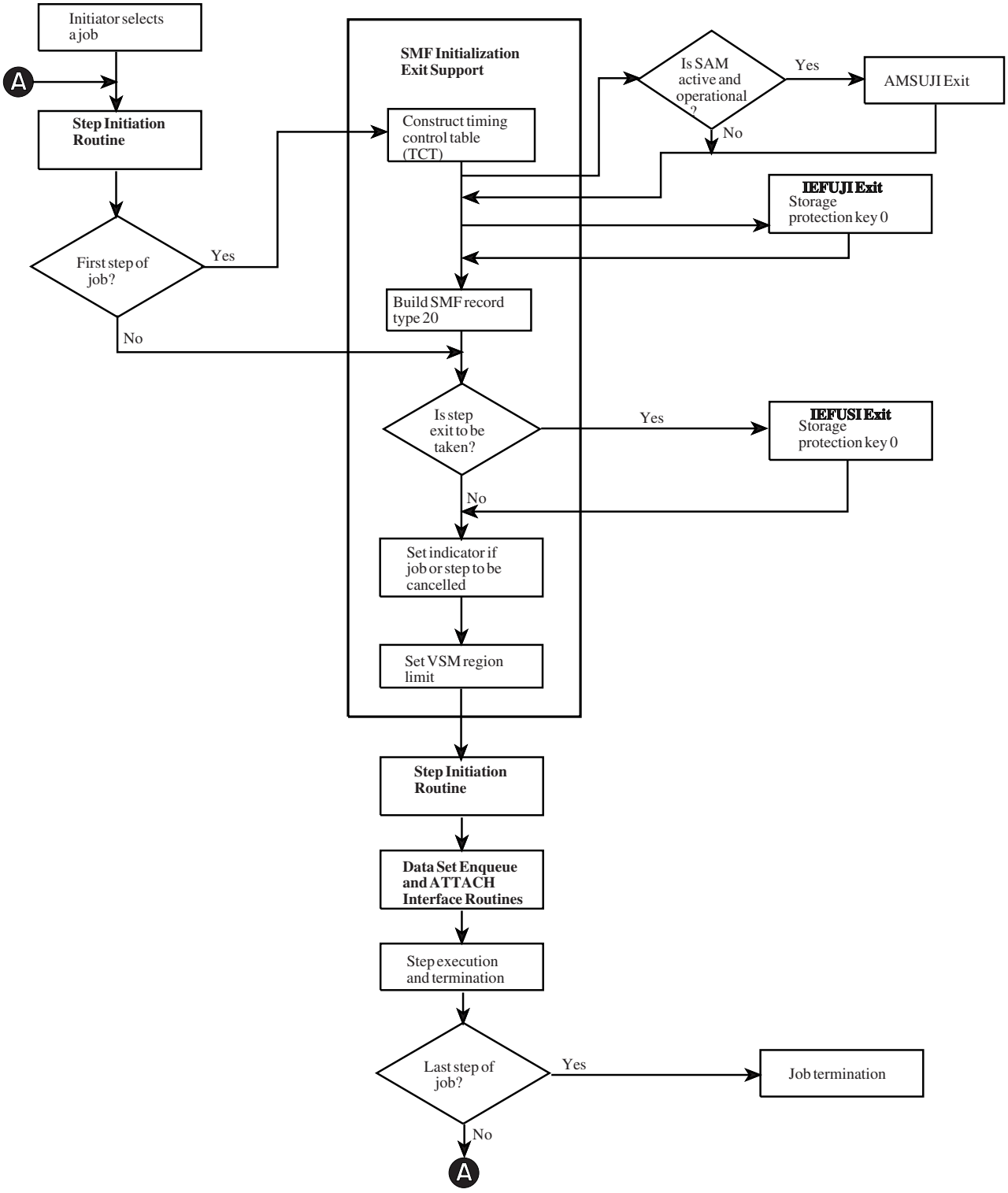

Figure 33. IEFUJI — Job Initiation Exit and IEFUSI — Step Initiation Exit

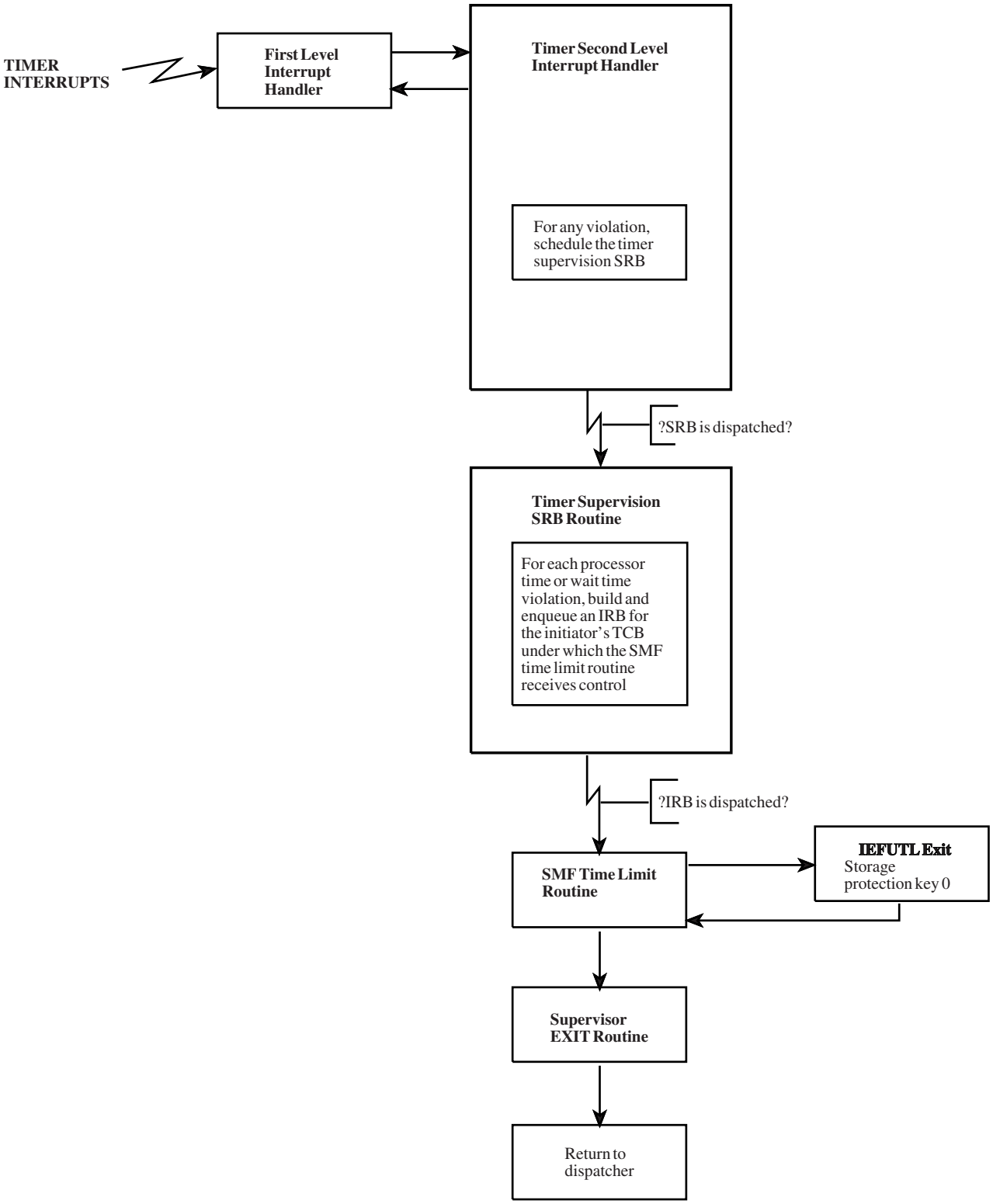

Figure 34. IEFUTL — Time Limit Exit

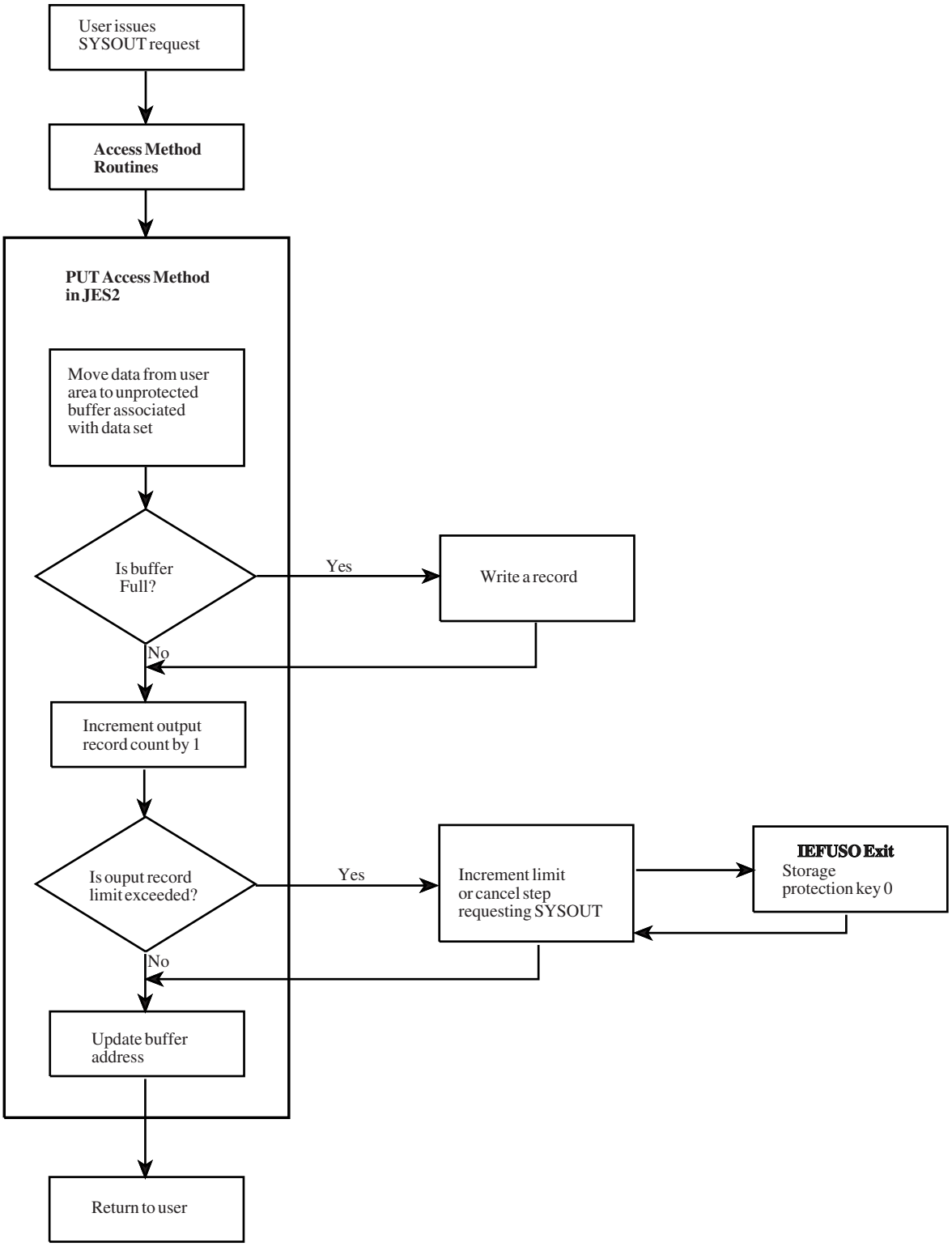

Figure 35. IEFUSO — JES2 SYSOUT Limit Exit

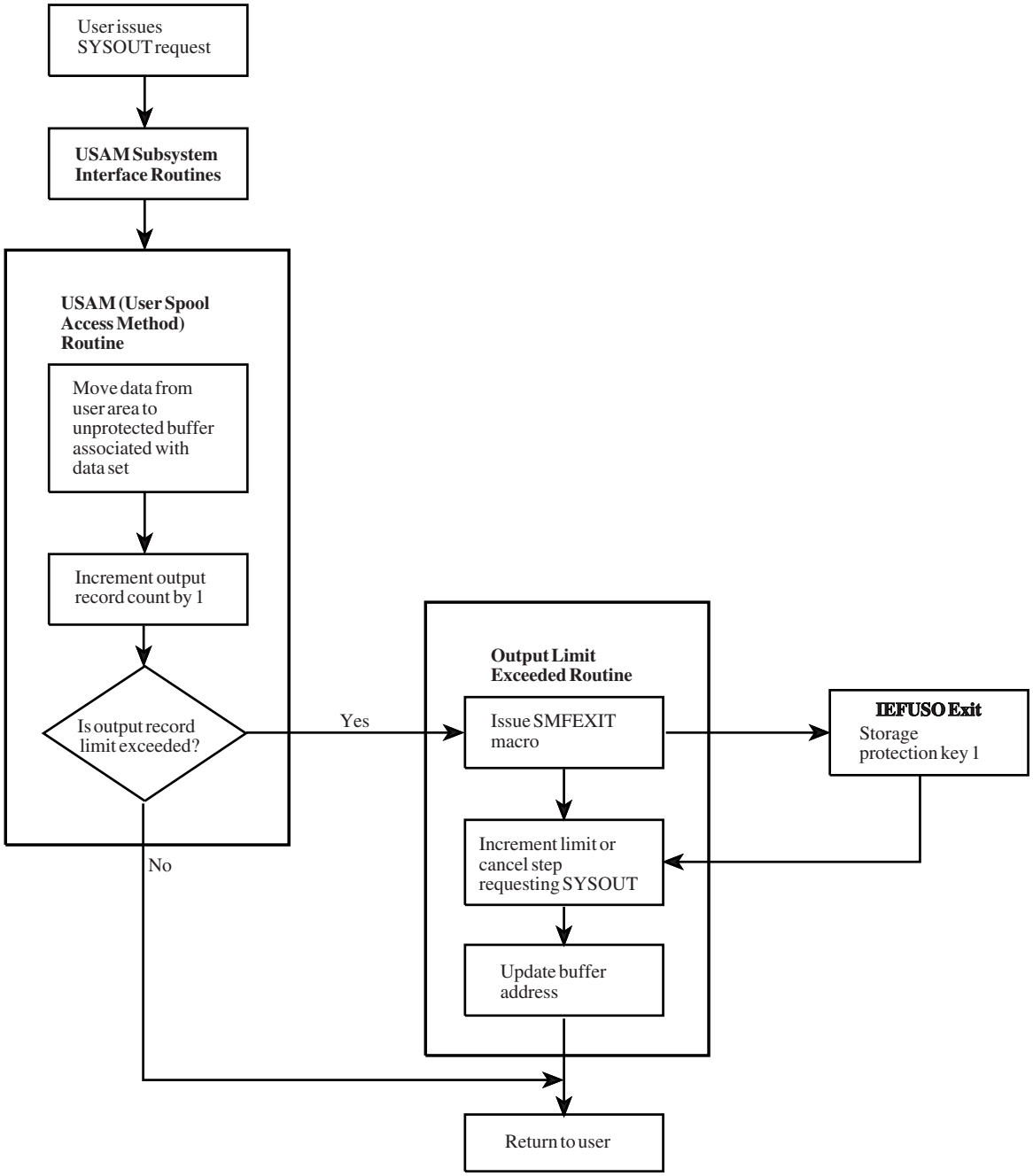

Figure 36. IEFUSO — JES3 SYSOUT Limit Exit

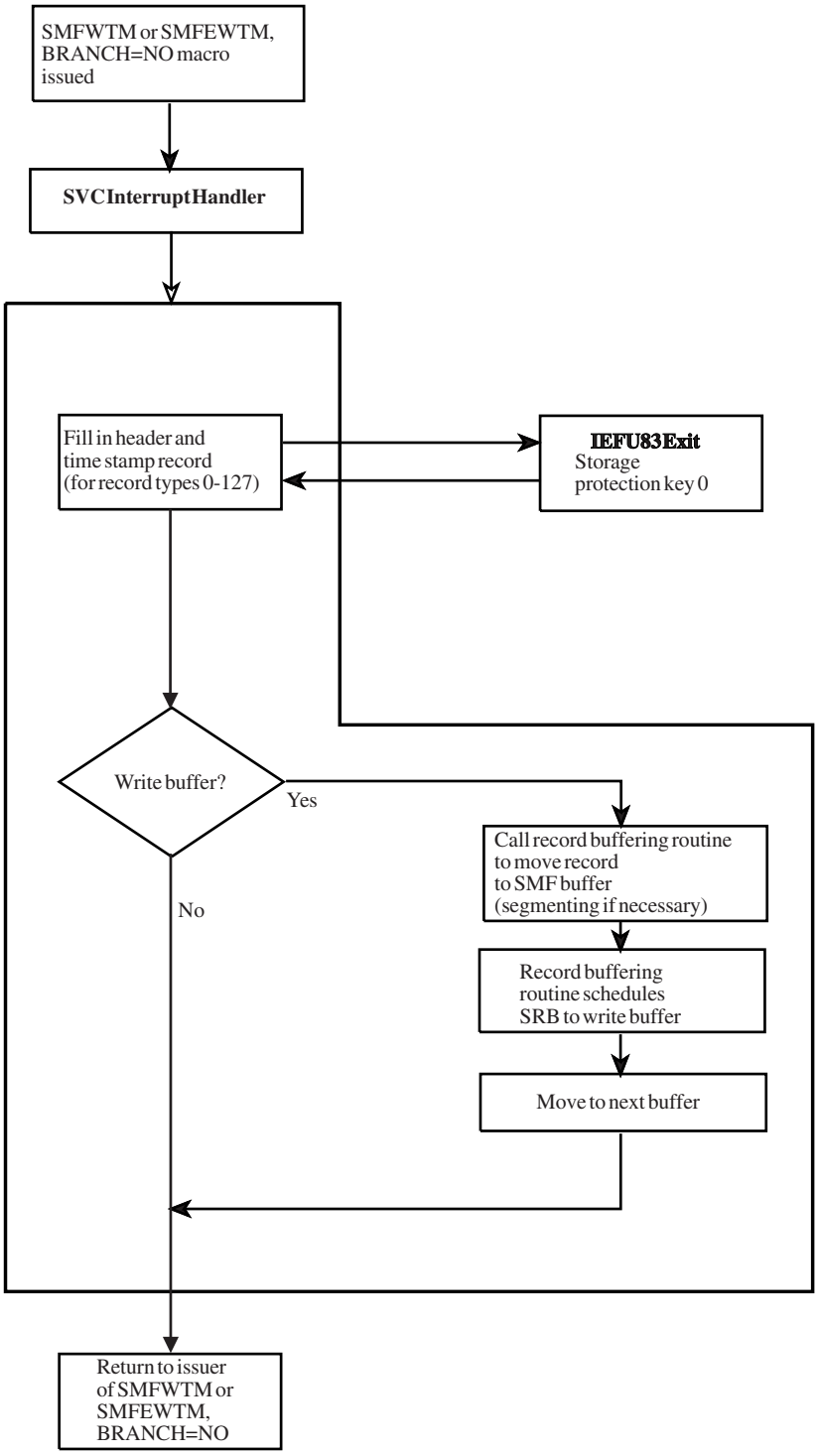

Figure 37. IEFU83 — SMF Record Exit

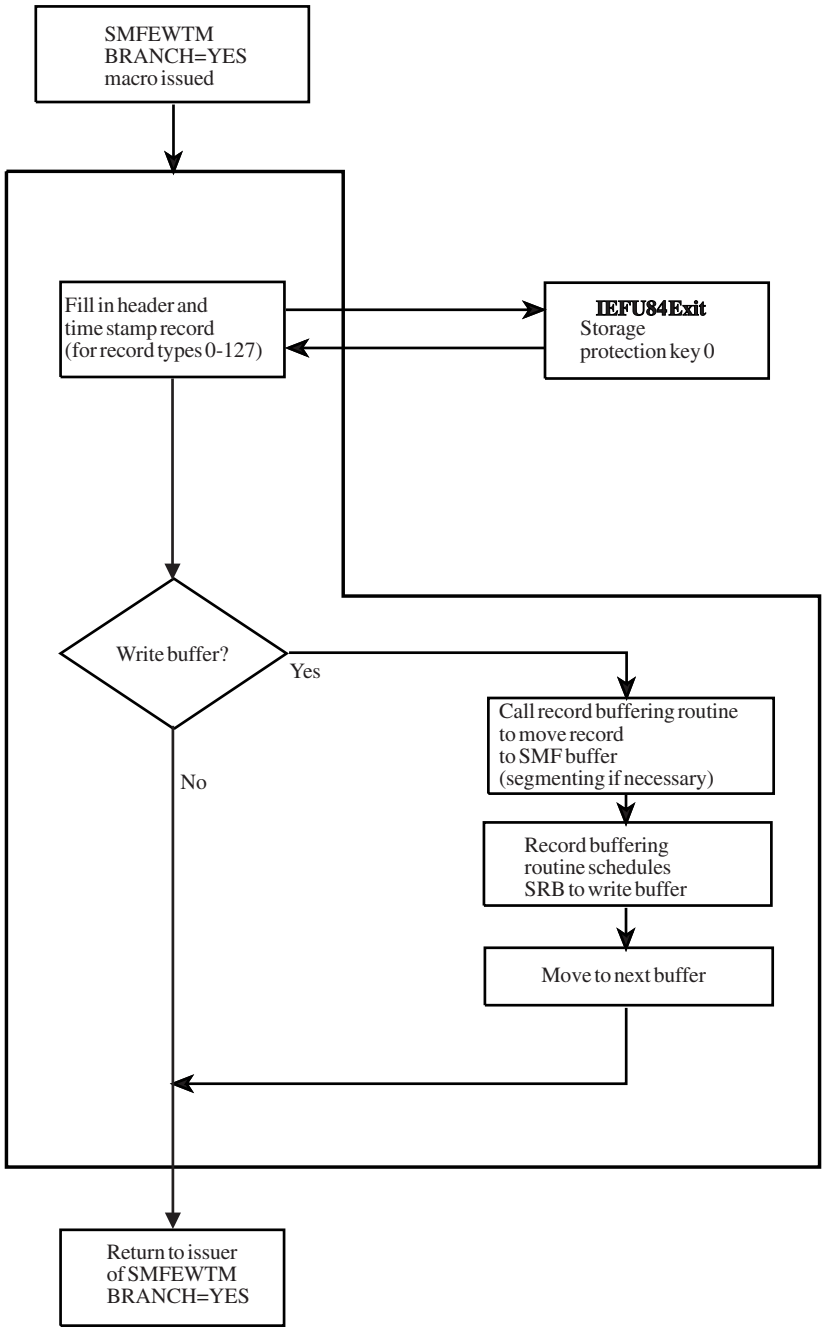

Figure 38. IEFU84 — SMF Record Exit

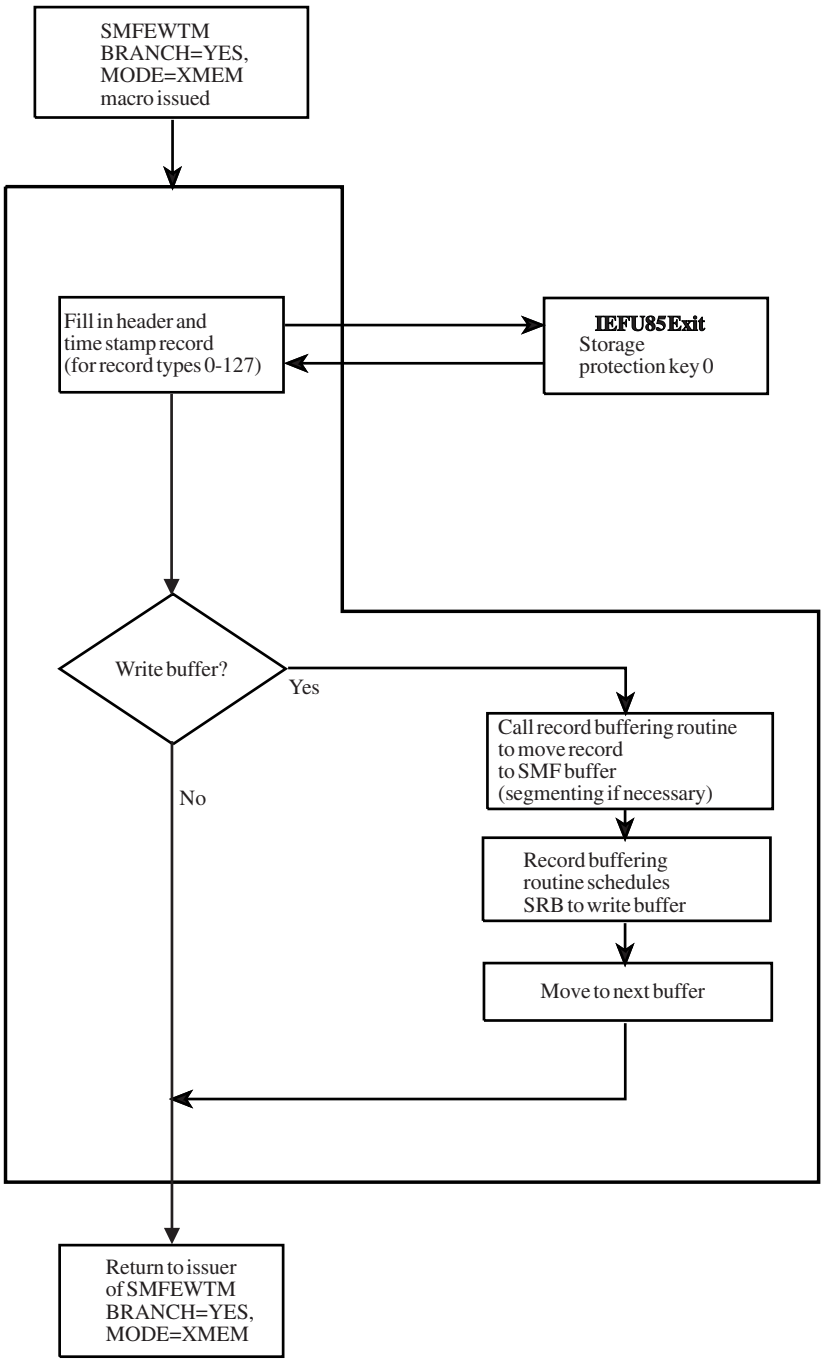

Figure 39. IEFU85 — SMF Record Exit

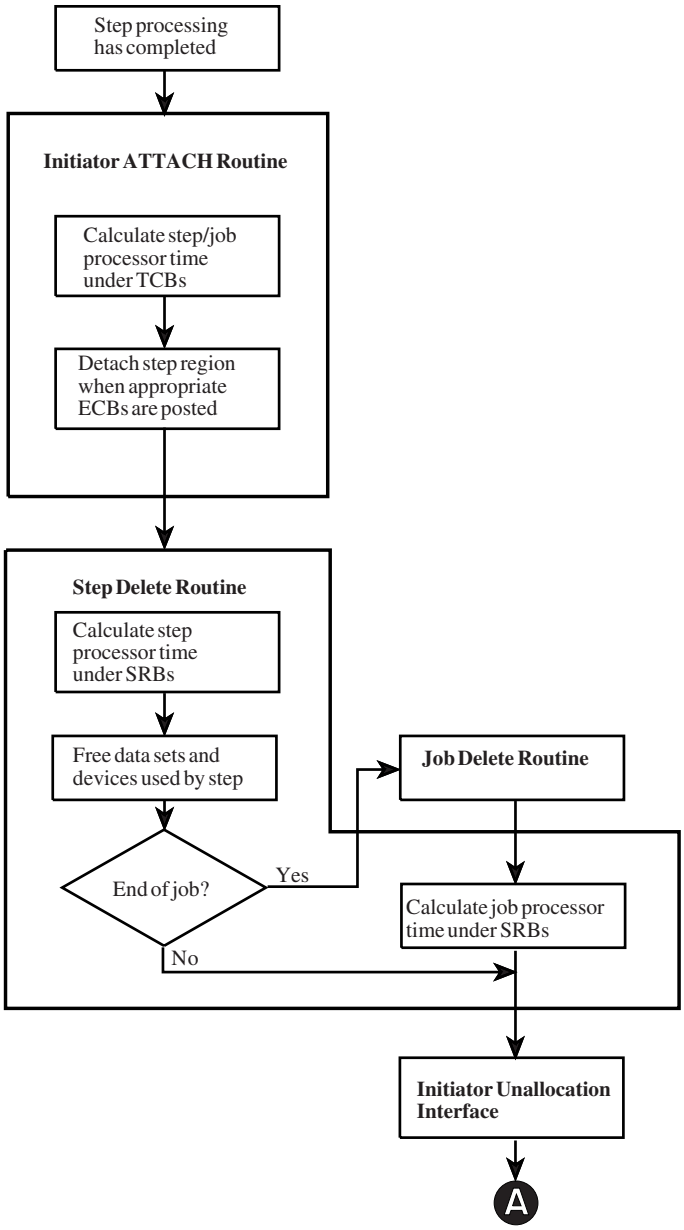

Figure 40. IEFACTRT — Termination Exit Part 1 of 2

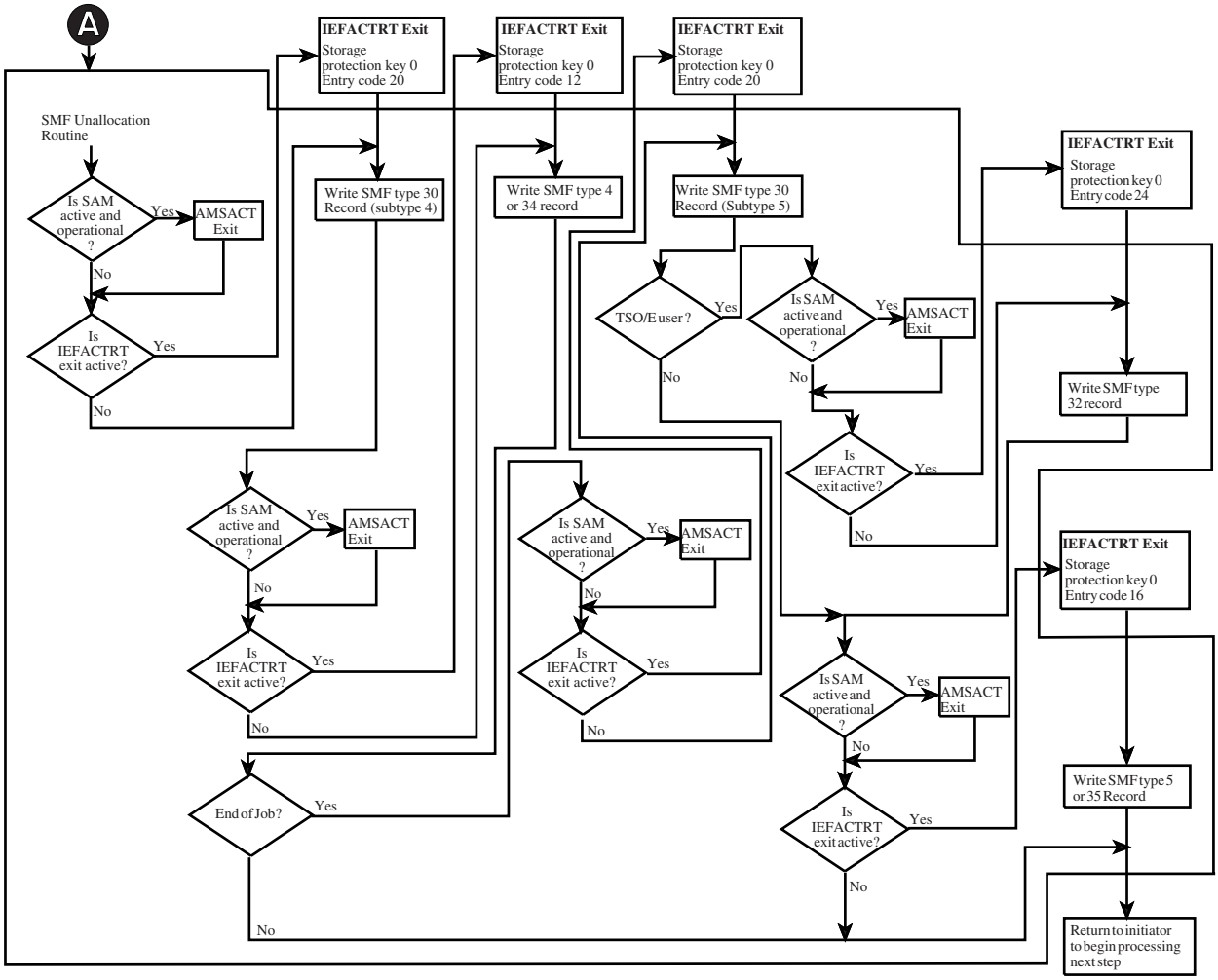

Figure 41. IEFACTRT — Termination Exit Part 2 of 2

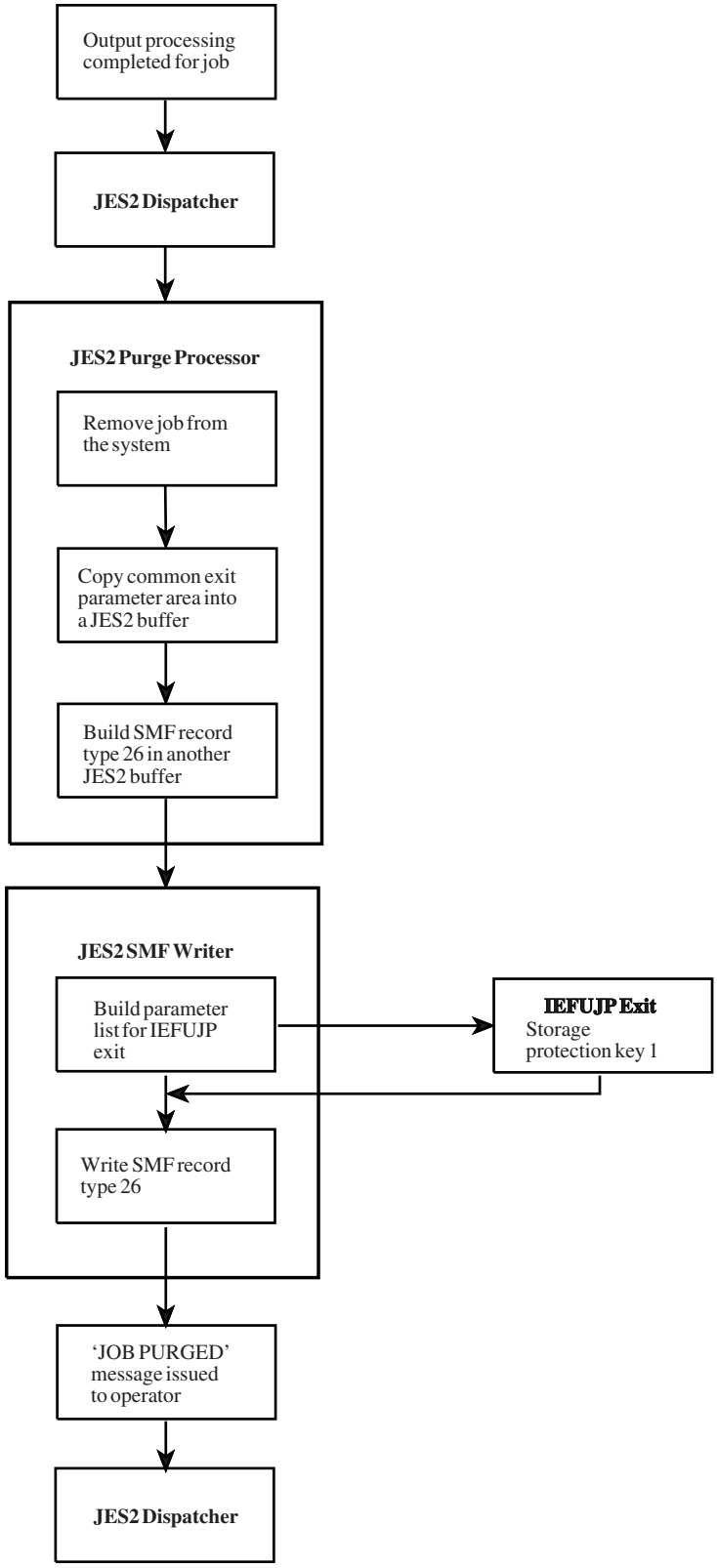

Figure 42. IEFUJP — JES2 Job Purge Exit

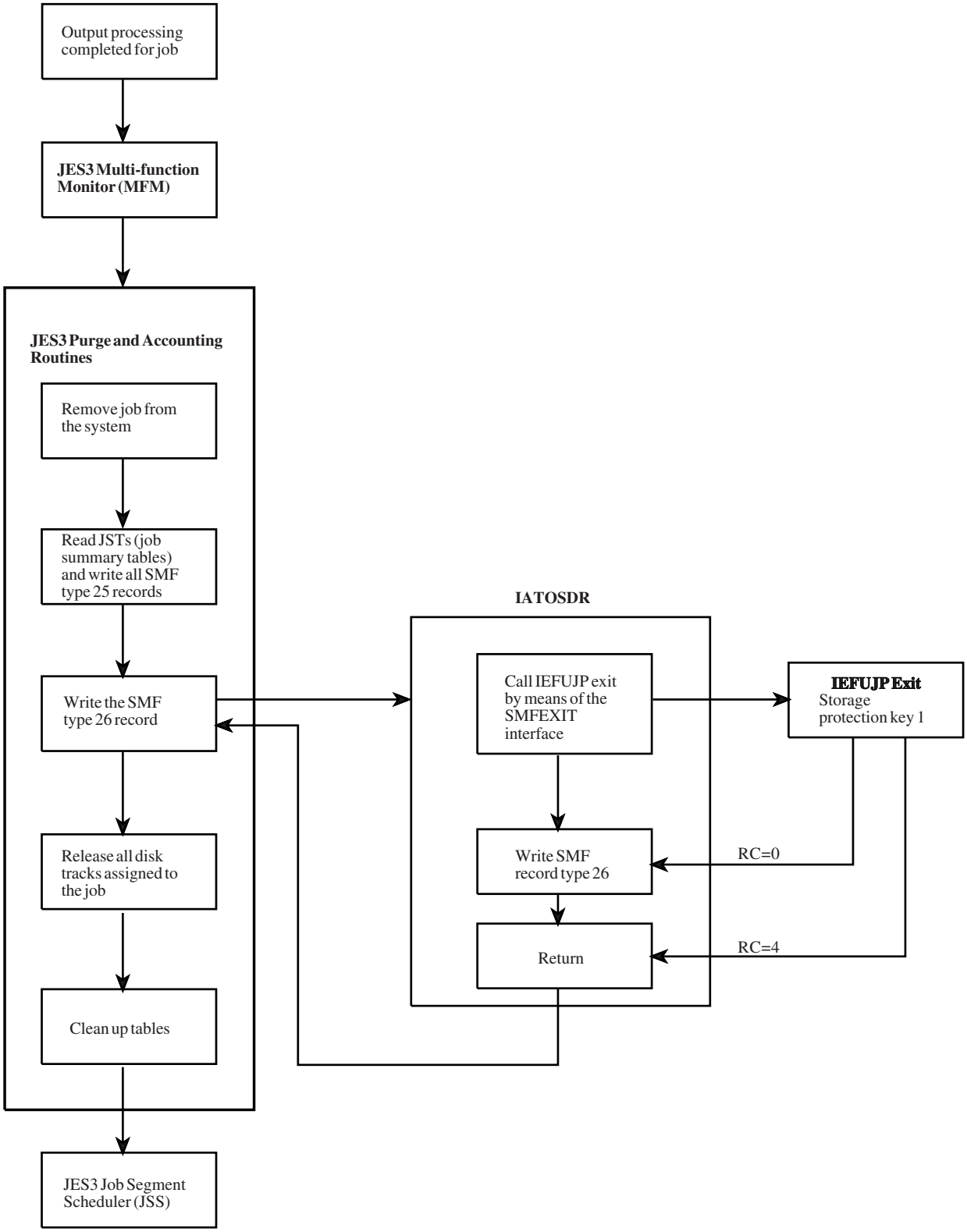

Figure 43. IEFUJP — JES3 Job Purge Exit

Part 7. Appendixes

Appendix. Accessibility

Accessible publications for this product are offered through the z/OS Information Center, which is available at www.ibm.com/systems/z/os/zos/bkserv/.

If you experience difficulty with the accessibility of any z/OS information, please send a detailed message to mhvrcfs@us.ibm.com or to the following mailing address:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size.

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the z/OS Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually

exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.
- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1!

(KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

- * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

  Note:

  1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
  2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
  3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte 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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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 wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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

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

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

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

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain 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.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted

for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming Interface Information

This manual is intended to help a customer modify the processing of an MVS operating system. This manual also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/OS.

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 (http://www.ibm.com/legal/copytrade.shtml).

Index

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Product Number: 5650-ZOS

Printed in USA

SA23-1381-00